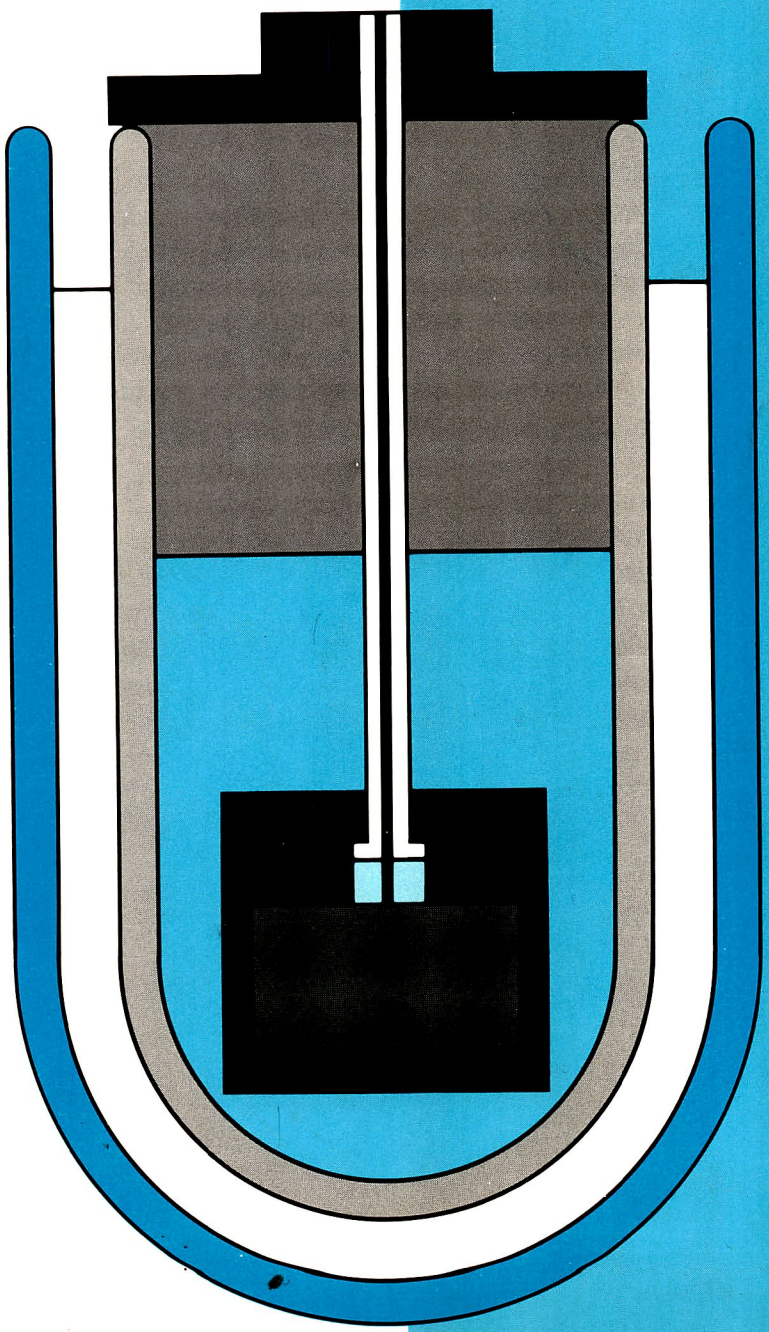
Wilbert N. Hubin

BASIC Programming For Scientists and Engineers



BASIC

Programming

for Scientists and Engineers

WILBERT N. HUBIN

Kent State University

PRENTICE-HALL, INC., *Englewood Cliffs, New Jersey, 07632*

Library of Congress Cataloging in Publication Data

HUBIN, WILBERT N

BASIC programming for scientists and engineers.

Includes index.

1. Basic (Computer program language) 2. Science—Data processing. 3. Engineering—Data processing. I. Title.

QA76.73.B3H82 001.6'424 77-21343

ISBN 0-13-066480-4

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

Contents

Preface

The computer is being woven more and more into our daily lives. Although much of this growth is in data processing, the computer has also become an essential and commonplace tool for the practitioner of science. A knowledge of what the computer can do and an ability to make it do it has become a necessary part of a technical education.

Thus this book presents a science-oriented introduction to the BASIC computer language; it also presents data analysis techniques and meaningful application problems for those who wish to develop additional programming proficiency in BASIC or some other computer language. A working knowledge of algebra and trigonometry is assumed; also, the formulas that are used for practice programs in Chap. I will be more meaningful to one who has taken an introductory physics course. Parts of Chap. III and problems complementary to it in Chap. IV assume some knowledge of calculus, but these sections are easily omitted.

Chapter I introduces the fundamental elements of BASIC through the use of description, review questions, and numerous short programming exercises. The exercises utilize the common formulas of science, and representative answers are provided. This is the only chapter that is language or machine dependent.

Chapter II discusses the flowchart method for clarifying and documenting the logic of a computer program. Basic symbolism is defined, and numerous examples and practice charting problems are given. This chapter provides a bridge to Chap. III for those who already know BASIC or another problem-oriented computer language.

Chapter III samples various techniques of numerical analysis. The emphasis is on methods that are relatively easy to understand but are still powerful and efficient enough for many scientific applications.

With the attitude that computers are lovable for their computing power, Chap. IV presents a variety of problems that aim to develop programming proficiency while demonstrating things the computer can do in many areas of science and engineering. The problems have been selected for their physical interest, general simplicity in concept, and a degree of computational difficulty that invites the use of the computer.

Chapters I and II and selected problems from Chap. IV have been the basis for an introductory course; Chap. III and selected problems from Chap. IV have been used for an advanced course. The inclusion of answers for all the problems should make the book particularly useful to the believer in self study.

The author gratefully acknowledges the assistance of Jean Schantz in proofreading, many thoughtful computer-oriented discussions with Chris Schantz, suggestions by Ron Ciminero for electronics problems, and the positive stimulus provided by students who have taken computer courses from him.

Kent, Ohio WILBERT N. HUBIN

BASIC

Programming

for Scientists and Engineers

I

Programming in BASIC

A. Introduction

The BASIC computer language is exceeded only by FORTRAN in popularity among scientific programmers. Nearly as powerful, BASIC is much easier to learn and use than FORTRAN. It obtains this relative ease by (a) making no apparent distinction between integers (1, 46, 4000, etc.) and numbers with fractional parts (1.0, 46.23, 4×10^{12}, etc.), and by (b) greatly simplifying the printing or displaying of calculational results.

BASIC (Beginner's All-Purpose Symbolic Instructional Code) was developed at Dartmouth College during the period from 1964 to 1966. It was developed for, and is particularly well suited to, a time-sharing environment. In this mode the computer can be accessed from numerous terminals; its internal memory has space allocated for each of the terminals, and it switches its attention sequentially among the terminals being used. Because input and output on the terminals is usually a very slow operation relative to the time required for calculations the programmer may not even be aware that he does not enjoy the undivided attention of the computer. Interaction between the computer and the programmer can be on a conversational level: The computer pauses to ask a question or receive a command and then hurries on as soon as the programmer has finished typing his message. Extremely powerful versions of BASIC have been and are being developed, because terminal access and the interactional mode are so useful in science and business applications of the computer. The proliferation of microcomputer and minicomputer installations will lead to an increase in the number of people learning about programming through the BASIC language.

The simplicity and power of time-shared computer programming in BASIC have made it very attractive for use by other than professional programmers. The scientist or engineer can use it to quickly make short calculations or analyze his data; more and more he is calling on the computer to monitor, record, and even control his experiments. The architect can make rapid estimates of building costs for various space, equipment, heating, and lighting options. The businessman can rapidly calculate principal or interest amounts and access data banks. The primary goal in learning BASIC, then, is to permit you to face the

computer as more than an equal; once you understand how the computer works and have had some experience in using it, application of its particular potentialities to your own problems is both possible and probable.

There is a fundamental difference between FORTRAN and BASIC that makes BASIC easier to learn and use but less suitable and less efficient for programs that do a very large amount of "number-crunching" or for programs that are to be run many times. When you tell the computer to begin the execution of a BASIC program, the BASIC interpreter in resident memory translates your statements into machine language (changing to the binary number system, moving numbers between memory locations, doing binary arithmetic, etc.) *as it executes the program*. However, with current versions of FORTRAN, the *complete program* is translated into an efficient machine language program by a FORTRAN compiler *before* execution is even attempted; once a program runs properly, only the efficient *translated* (compiled) program is used.

Yet in many applications, the efficiency of the final calculation is not as important as the programmer and computer time required to obtain a working program. In this aspect BASIC and time-shared terminals are superb; typographical errors can be corrected in seconds rather than hours, and the machine can be immediately quizzed for a further explanation of questionable answers or of error messages. Also, in some versions of BASIC, the computer translates each *line* of a program as it is being entered, which greatly enhances execution efficiency.

B. Mathematical Operators, Error Correction, and Accuracy

Most versions of BASIC allow the programmer to use the computer as a calculator (this is sometimes referred to as the *immediate mode*.) It means that short, single-answer (but possibly very complex) calculations can be made *without* writing a computer program. [This and the next section assume that your system has this capability. If it does not, you need to type a number (for example, **10**) at the beginning of the line and then type a **RUN** command after the desired calculation and a **RETURN** key have been typed.]

In making friends with the computer, the first step is to get it to recognize you. This may be as simple as turning on the terminal or as complex as specifying the language to be used and supplying a valid ID and password, depending upon the particular computer and installation. (See Appendix II for a sample log-on, log-off procedure.)

In the simplest use of the immediate mode, the programmer types the command **PRINT** followed by (a) the characters to be retyped enclosed in quotes, and/or (b) the numerical expression to be evaluated, using the mathematical operators described in the following.

Whenever two quote marks appear in a PRINT statement, the computer prints out anything and everything appearing between the quote marks, without trying to understand or interpret it. For example, typing

```
PRINT "NUMBER 306-4973-123, I LIKE YOU."
```

followed by a typing of the RETURN key would cause the message

```
NUMBER 306-4973-123, I LIKE YOU.
```

to be printed.

The arithmetic operator symbols in BASIC are +, –, *, /, and ↑. The first four symbolize requests, respectively, to add, subtract, multiply, or divide two numbers. The last symbol, ↑, is the exponentiation operator. Thus the typing of

```
PRINT 2↑3
```

is interpreted as a command to calculate and print out the quantity 2^3, and the interpreter returns with an answer of 8.

Before the days of electronic calculators, an expression such as $(3.718)^{2.5}$ was evaluated by taking the logarithm of 3.718, multiplying it by 2.5, and then finding the antilogarithm of that number. The BASIC interpreter does it in much the same way, although some are smart enough to simply multiply the number by itself if the exponent is an integer. But most interpreters choke up when asked to evaluate an expression such as (–2)↑3 because the logarithm of a negative number is not defined as a real number; an error message is its reply and your reward. It is safer and generally more efficient to use the multiplication operator when a quantity is to be squared or cubed.

The computer uses a power series approximation to calculate such things as logarithms or trigonometric functions. This illustrates the aptness of the common appellation of "high-speed moron" to the computer; it can only store and print characters, add, subtract, multiply, and divide–but it can do it very rapidly!

Most versions of BASIC have the following designed-in mathematical functions:

GENERAL

ABS(X) –returns the absolute value of any number represented by the symbol X.

INT(X) –returns the greatest integer less than or equal to X.

RND(X) –returns a random number between 0 and 1 for any X.

SGN(X) –returns a 1 if X is a positive number, 0 if X is zero, and –1 if X is a negative number.

PI –returns an approximate value for π, 3.14159 . . .

SQR(X) –returns the square root of a positive number X.

TRIGONOMETRIC

ATN(X) –returns the angle in *radians* whose tangent is the number X. The angle is assumed to be in the first quadrant (0° to 90°) for positive X and in the *fourth* quadrant (0° to –90°) for negative X.

COS(X) –returns the cosine of the number X, assuming X to be an angle expressed in *radians*.

SIN(X) –returns the sine of the number X, assuming X to be an angle expressed in *radians*.

EXPONENTIAL

EXP(X) –returns the value of e raised to the X power $\equiv e^X$, where $e = 2.71828\ldots$.

LOG(X) –returns the natural logarithm (base e) of the number X.

In the preceding list, X represents either a number *or* any BASIC expression that can be evaluated to obtain a number, such as another function.

Notice that the trigonometric functions require that the angle be expressed in radians; the conversion factor is $\pi/180$ so that, for example, we could determine the sine of 30° with the command

```
PRINT SIN(30*PI/180)
```

which would return the expected value of 0.5. (However, if your BASIC does not have the PI function, then you will have to supply it. You should give it to as many significant figures as your machine can handle; don't tell the interpreter that π is equal to 3.14 and then blame me when your answers differ from mine! Also, these trigonometric functions may not have full accuracy over their whole defined range.)

The computer statements

```
PRINT 9↑.5
```

and

```
PRINT SQR(9)
```

both return a value of 3 on the printer, but the second form takes the computer only about half as long to execute as the first form because the square root can be obtained in a *single* power series. Therefore, it is considered good programming practice to use this square root function whenever applicable.

Scientific notation is understood and used by the BASIC interpreter. Powers of ten are written using the letter E because neither small letters nor subscripts are generally available on the keyboard. Thus the division $(4 \times 10^{10})/(2 \times 10^{-10})$ can be written as

```
PRINT 4E10/2E-10
```

and the interpreter returns a value of 2.00000E+20.

When the computer is printing out answers, it uses a fixed decimal point if no significant figures are thereby lost. Otherwise, it automatically switches to the exponential or scientific form. The largest and smallest numbers that you can give to the computer or have calculated by it are about 10^{38} and 10^{-38} respectively, although this varies with the particular installation. It is important to understand that typing 4↑12 is asking the computer to perform the calculation 4^{12}, whereas typing 4E12 is simply giving the computer the number 4×10^{12}.

RND(X) is a particularly useful function for doing statistical simulations or for playing games on the computer. One use for the SGN(X) function is illustrated in Chap. IV in the discussion of the solution of a cubic equation.

If you have really messed up the typing of a statement or have made an error at the beginning of the line, you may wish to give up and retype the whole line. BASIC always provides a method for making this withdrawal a graceful one. In some versions, this is done by typing CTRL/U, which means that the key labeled U is typed while holding down the key labeled CTRL. Of course you can always give up by typing the RETURN key, but that will get you an incorrect answer at best or an error message at worst.

If you only want to correct part of a line, usually the RUBOUT key is the one you want. In installations that use this, each use of this key erases one previous character from the computer memory, and a backwards arrow, ←, may be printed each time to tell you how many previous characters have been expunged. For example, if you wanted to type

```
PRINT 4.628*7.423
```

but instead typed

```
PRINT 4.628*7.623
```

you could save the day (or at least the moment) by typing RUBOUT three times and then typing 423. The final line would look like this on the printer:

```
PRINT 4.628*7.623←←←423
```

but the computer has both forgiven and forgotten and would store only the corrected version.

Most versions of BASIC are accurate to at most seven digits, and the printer prints one less digit than that. However, more and more interpreters that make extended precision a programmer option are becoming available. It is a particularly useful option for engineering applications because many calculations involve the manipulation of rounded-off numbers, but it does require additional internal memory for both the interpreter and for number storage, plus it reduces execution speed. Yet few among us are ever going to take data with more than six significant figures, so normal BASIC is sufficient as long as errors aren't allowed to grow and multiply.

In usual practice the subtraction of nearly equal numbers and the accumulation of round-off errors are responsible if the calculated answer is accurate to fewer than six digits. Consider the calculation

```
PRINT 794.32-794.31
```

The answer given by my computer is 9.94873E-03, or 0.00994873! (How stupid does it think we are!) The numbers subtracted each contain *five* significant figures, but the answer is meaningful to only *one*. (The reason the computer didn't return a value of 01 is that it uses *binary* arithmetic; that is, a number system based on *two* rather than ten. There is in fact no exact representation for .01 in a base 2 number system.)

The computer rounds off its calculated values to about seven significant figures; but in a series containing many terms, the accuracy depends on the order of the terms, and the accumulated error can be quite significant. For example,

```
PRINT 1000001-1000000-.4
```

gives an answer of .6, but

```
PRINT 1000001-.4-1000000
```

returns a much less accurate .625, because the operations are performed in sequential order from the left to the right. One general method to fight the problem is to group the numbers according to their absolute order of magnitude, do the additions and subtractions within each group, and then repeat the process until all the additions and subtractions have been completed. However, the additional manipulations may introduce their own round-off errors to the extent that gains are nullified.

Enough talk; it's high time you began your career with BASIC! Verifying the practice calculations given at the end of this section will help to familiarize you with the arithmetic functions and give you confidence in dealing with the machine. If you make a typing error that you don't correct, or if you don't have the same number of right and left parentheses, then you can expect to receive an error message such as **?SYN** rather than a numerical reply. ?SYN is an abbreviation for SYNTAX ERROR and means that you have strayed outside the vocabulary of the BASIC interpreter. So look at your work carefully and try again, and please try to avoid calling the machine any nasty names–remember Hal. (Could your computer be listening, too?)

Practice Calculations

1. $7.46 * 8.19 = 61.0974$
2. $4.68 + \frac{7.17}{1.32} = 10.1118$
3. $\frac{2.19}{.056} - 40.3 = -1.19286$
4. $7.3^7 = 1.10474E+06$
5. $1.862^5 = 22.382$
6. $\sqrt[3]{14.68} \equiv 14.68^{1/3} \equiv 14.68\uparrow(1/3) = 2.44855$
7. $7.69^{1/3} = 1.97383$
8. $pi = 3.14159$
9. $\sin(1.67) = .995083$
10. $\tan(0.68) \equiv \sin(0.68)/\cos(0.68) = .808661$
11. $\sin(45°) = .707107$
12. $\cos(130°) = -.642788$
13. $\tan^{-1}(-1) \equiv \text{arc tan}(-1) \equiv \text{ATN}(-1) = -.785398$ (radians)
14. $\tan^{-1}(1) = .785398$
15. $\tan^{-1}(1.784) = 1.0599$
16. $\sqrt{9} \equiv \text{SQR}(9) = 3$
17. $\sqrt{4.678} = 2.16287$
18. $e \equiv e^1 \equiv \text{EXP}(1) = 2.71828$
19. $e^{4.673} = 107.018$

20. $\log_e (2.7) \equiv \ln (2.7) = \log (2.7) = .993252$
21. $\ln (87.63) = 4.47312$
22. $\ln (e^1) = 1$
23. $2 * \pi * (3)^2 = 56.5487$
24. $|-1.4| \equiv$ ABS $(-1.4) = 1.4$
25. INT $(4.6) = 4$
26. INT $(-4.6) = -5$
27. RND $(7.3) =$ (random!)
28. SGN $(4.6) = 1$
29. SGN $(4.6 - 2 * 2.3) = 0$
30. SGN $(-4.6) = -1$

C. The Correct and Efficient Use of Parentheses

No two arithmetic operators can appear directly next to each other. Thus the expression 4*−5 is not understood and not accepted by the BASIC interpreter. Parentheses are used to show the dumb machine that you merely wanted to designate 5 as a negative number: 4*(−5).

BASIC normally does a calculation sequentially from left to right; thus

```
7.4 + 4.6/2
```

is interpreted as a request to calculate

$$7.4 + \frac{4.6}{2}$$

rather than

$$\frac{7.4 + 4.6}{2}$$

To obtain the last calculation, parentheses can be used as in

```
(7.4 + 4.6)/2
```

Whenever parentheses are encountered, the machine does the operations specified within the *innermost* set of parentheses first and then works its way outward. For example, in performing the calculation

```
2•(4 + 3•(7/2))
```

the machine (a) divides 7 by 2, then (b) multiplies the result by 3, then (c) adds that result to 4, and finally (d) multiplies that sum by 2. In Appendix I there is a listing of the priorities for the various mathematical operations. These priorities indicate the order in which these operations are performed when they occur in the same expression in the *absence* of parentheses. Thus in the first example above, 4.6 was divided by 2 *before* it was added to 7.4, because division has a higher priority than addition.

Similarly, in the expression 7*6↑3, the calculation is of 7×6^3 rather than $(7 \times 6)^3$, because exponentiation has a higher priority than multiplication.

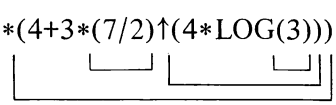
Some versions of BASIC interpret $-4\uparrow2$ as $-(4)^2 = -16$, but other versions interpret it as $(-4)^2 = +16$. This should warn you of the importance of careful use of parentheses when the negative sign is combined with exponentiation: $-(4\uparrow2)$ is unambiguously equal to +16 in both versions. Computers are world renowned for an implacable attachment to their own peculiar brand of logic.

Multiplications and divisions have the same priority, so they are accomplished from left to right in the absence of parentheses. Thus the calculation $(4 \times 7)/(3^5)$ can be written in *any* of the following forms:

1. ((4*7)/(3↑5))
2. (4*7)/(3↑5)
3. 4*7/(3↑5)
4. 4*7/3↑5
5. 4*7/3/3/3/3/3

Among these, I suggest to you that form 3 is the best. Although form 4 is simplest, parentheses would be needed in the denominator if any operation besides exponentiation were occurring. Forms 1 and 2 represent "overkill" with parentheses. Form 5 is somewhat confusing to read.

Because an odd number of parenthesis marks represents a logical ambiguity, the interpreter will stop to ask you what you really wanted, with something like the ?SYN message. Parentheses can be checked by drawing lines under a statement and thus grouping them from inner to outer:



Discreet factoring and parentheses can sometimes be used to reduce the number of calculations that the machine has to make, and this can be significant if the calculation has to be performed many times (as in a loop). For example, the expression

$$1 + 2*3 + 4*3^2 + 5*3^3$$

requires three additions, three multiplcations, and two exponentiations to be evaluated. But writing it in the form

```
1 + 3*(2 + 3 * (4 + 5*3))
```

reduces the number of operations to three additions and three multiplications.

You may be fearful at this point that multiple groupings of parentheses will lead to ulcers. Take heart; we see later that in actual programs, parentheses can be greatly reduced in number, or even eliminated, and clarity greatly enhanced by doing a calculation in more than one step. Carefully study the statements and examples of this section so that you can use parentheses both efficiently and correctly. You will be rewarded with a net saving in time.

Review Questions: Find the errors in the following immediate mode expressions.

1. `PRINT (1.8)(3.2)`
2. `PRINT SIN(3.4*-6.1)`
3. `PRINT (3/(4+6)/SIN(.3)`
4. `PRINT LOG4.6-2COS(.6)`
5. `PRINT (3.6-(7/.3))↑.6`
6. `PI/7(6-3*ATN(3)`

Practice Calculations: It is recommended that you write out these expressions in the form that you would type them *before* you get to the terminal. Then brace yourself for a little bit of back talk, particularly if your typing ability is less than professional, but use the correction key as necessary.

1. $\dfrac{8 - 3}{2} = 2.5$
2. $\dfrac{1 - 8}{2 + 5} = -1$
3. $2\pi(3)^2 = 56.5487$
4. $\dfrac{4.32 - 9.62}{1.83} = -2.89618$
5. $\dfrac{1.97 + 4.23}{6.72 - 3.18} = 1.75141$
6. $(7.6 \times 10^{12})(3.4 \times 10^{-20}) = 2.58400\text{E}{-07}$
7. $(-8.37 \times 10^{-12})[(4.11 \times 10^{3}) - (9.23 \times 10^{3})] = 4.28544\text{E}{-08}$
8. $2.39(1.093 \times 10^{20})^{1/2} = 2.49866\text{E}{+10}$
9. $e^{-4.3249} + \log_e (9.127 \times 10^{19}) = 45.9736$

10. $(3^2 + 4^2)^{1/2} = 5$

11. $[4.621 + (6.324)^{1/2}]^4 = 2592.75$

12. $\dfrac{7.8[1 + (3/7)]}{6.1 + 3 \cdot 1.4^4} = .632226$

13. $|4.3 - 7.8[1 + (5/3)]| = 16.5$

14. $4.728^{[3+(7.8/2.3)]} = 20514.9$

15. $\dfrac{4.6 \log_e \left[\dfrac{7}{12 + (1/3)}\right]}{1 + e^{(4.6+.73^4)}} = -.0195667$

16. $\dfrac{7 + \dfrac{4}{6 + (7/3)}}{14 + \dfrac{2}{6^{7/8}}} = .518832$

D. The LET Assignment Statement, The PRINT Statement, and Program Commands

Thus far you have worked only with constants, that is, various specific numbers. Often, though, we wish to make a certain calculation over again with different numbers. In fact, the ability to store a sequence of operations before assigning numbers to the operands is where a calculator stops and a computer begins. The key to this ability, as in normal scientific practice, is to represent the calculation by symbols or *variables* that can take on any value in some allowed range. Consider the expression for distance traveled by a vehicle uniformly accelerating from rest:

$$s = \tfrac{1}{2}at^2$$

where s is the distance traveled after accelerating at a rate a for a time t. A complete BASIC program that would make this calculation for $a = 20$ and $t = 5$ is

```
10 LET A=20
20 LET T=5
30 LET S=A*T*T/2
40 PRINT S
50 END
```

What does the statement **LET A=20** mean? It means that we are instructing the BASIC interpreter to take the number **20** to a location of its own choosing and to label the address of that location with our symbol **A**. Therefore the LET statement is an *assignment* state-

ment: The number 20 is assigned to the symbol **A**. Sometimes a backwards arrow is used to emphasize this:

```
A ← 20
```

Note that all the statements in the BASIC program are numbered and that the END statement is the highest-numbered statement; the presence of the numbers tells the interpreter that you are using it in the program mode and will separately command it to begin execution. (Statements may be numbered with integer numbers between 1 and some larger number such as 32767, depending upon the version of BASIC.) The statements are always executed *sequentially*, that is in numerical order, *unless* specific instructions to the contrary are contained in the program (see Sec. H). The numbers that you use are arbitrary, but it is customary to use a spacing of ten to make it easy to insert additional statements at a later time. They are always stored, listed, and executed sequentially, independent of the order of entry.

It is of great importance that you understand the distinction between a location and its contents. In statement number 30, the command is to take the *number* stored in location **A**, multiply it twice by the *number* stored in location **T**, divide the result by 2, and finally store that *number* in a location whose address is **S**. If we change what is stored in locations **A** and **T**, then statement 30 will calculate and store a different value for **S**. But to make a meaningful calculation, we logically must have stored something at a location before we try to use it! If you try to run a program such as

```
10 LET S=4*T
20 LET T=1
30 PRINT S
40 END
```

the results will be meaningless at best and embarrassing at worst. Many computers, when faced with an "undefined variable," such as **T** in statement 10, "wander" through their internal memory and seemingly arbitrarily pick up a number, and using that number continue through the calculation as if nothing was amiss! In some other versions, this won't happen, because the BASIC interpreter sets all variables equal to zero when the RUN command is received. *However*, you are strongly urged to always set variables equal to zero *yourself* if they are to be used in a summing statement (as described in the next paragraph), lest most mysterious ills plague your future programs.

Statements such as LET N=N+1 are commonly seen in computer programs and illustrate again the greater appropriateness of the backward arrow: N←N+1 means to assign a new value, one greater than the previous one, to the number stored at location **N**.

There is a growing trend toward making the use of the word LET optional. Originally the word was used to emphasize to the beginning programmer that the assignment statement was not a normal algebraic equation. However, its use does require additional typing effort and does require additional memory for program storage. Thus in the interests of efficiency, LET is omitted in the remainder of this text. (Without the LET, numerical operations

written in BASIC look very much like their counterparts in FORTRAN, with the notable exception that FORTRAN uses two asterisks to indicate exponentiation.)

The program

```
10 A=20
20 T=5
30 A=A*T*T/2
40 PRINT A
50 END
```

returns the same value as the first program; however, it requires less memory because it uses one less storage location, although it suffers some loss in readability. Statement 30 now means to take the number stored in location **A**, multiply it twice by the number stored in location **T**, divide the result by 2, and finally store that result in location **A** (wiping out the 20 stored there by statement 10).

What are the permissible variables or variable names? In normal BASIC a numerical variable can be a single letter of the alphabet *or* a single letter followed by a single digit. Thus **A**, **X**, **Y**, **B0**, **Z1**, **Z2**, and **N5** are all allowable forms, but **AC**, **4N**, **B10**, and **QU1** are not. Usually variable names are chosen to suggest the quantity being represented, as in the previous example. (This is one instance in which FORTRAN is considerably more flexible than BASIC—it allows the use of five or more characters.)

A very useful feature of many versions of BASIC is the availability of multiple-statement lines. The symbol that is used to separate the different statements on a line is typically the back slash (\) or the colon (:). The back slash seems more readable to me and, therefore, is used hereafter. In any case, there is *no* way to continue a statement from one line to the next. With multiple-statement lines, our last program could just as well have been written as

```
10 A=20\ T=5
20 A=A*T*T/2\PRINT A
30 END
```

The advantages of using multiple-statement lines are (a) the compacting of the written program and (b) some saving in required memory. Disadvantages include (a) the need for more retyping when making corrections and (b) the inability to later refer to any statement in the line except for the first (see Sec. H).

We should note at this time that all spaces in a BASIC program (except for those between quotes in a PRINT statement) are optional. Ordinarily spaces are used to enhance readability, but excessive spacing wastes memory. Some interpreters make the decisions themselves as to where spaces should finally go, so that you might type in

```
20S=A*T*T/2\PRINTA
```

but it might be stored and later printed out as

```
20 S=A*T*T/2\ PRINT A
```

As a final comment on the assignment statement, consider the mathematical formula for the bending moment anywhere along an airplane wing that has uniform taper (see subsection C-5 in Chap. IV):

$$M = \frac{W \cdot (X - L)^2}{6 \cdot L^2 \cdot (C1 + C2)} \left([(C1 - C2) \cdot X] - \left\{ [C1 + (2 \cdot C2)] \cdot L \right\} \right)$$

This calculation could be accomplished by using a single LET statement (assuming we have previously assigned values to *W*, *L*, *X*, *C*1, and *C*2) as follows:

```
40 M=W*(X-L)*(X-L)*((C1-C2)*X-(C1+2*C2)*L)/(6*L*L*(C1+C2))
```

However, it is much easier to type correctly and to check, if it is written with fewer parentheses in a *series* of statements, like so:

```
40 M=(C1-C2)*X-(C1+2*C2)*L
50 M=M*W*(X-L)*(X-L)
60 M=M/(6*L*L*(C1+C2))
```

All of the sample programs so far have shown an END statement as the last statement. To some extent this is an unnecessary carry-over from other computer languages; it is used to signal the physical end of a program so that a compiler can recognize the end of a main program and the beginning of a subprogram. However, because a BASIC program is interpreted as it is being executed, it is natural for execution to stop when there are no more instructions, and you may find that use of the END statement is optional.

We see later that the printer can be sent anywhere on the output sheet, but for now we concentrate on the features of the PRINT statement, which will quickly give you the ability to obtain answers in a readable format.

Your output device (video screen or printer) most likely has 70 available positions on each line. BASIC normally divides these into 5 print zones of 14 spaces each. The statement

```
50 PRINT X,Y,Z
```

generates a command to the printer to print the number stored at location X in the first available position in the first print zone, then to move to the beginning of the second print zone to print the number stored at location Y, and finally to move to the beginning of the third print zone to print the value of Z. But the first comma in

```
50 PRINT ,X,,Y
```

causes the printer to skip the first print zone and begin printing at the beginning of the second print zone, and the number stored for Y is printed in the fourth print zone. And the comma after the **Y** in

```
60 PRINT X,Y,

70 PRINT Z
```

causes the printer to print the next number in the third print zone rather than moving on to the next line.

For large quantities of output, column headings are usually printed before the answers. For example,

```
10 PRINT "X","Y","Z"
```

prints **X**, **Y**, and **Z** in the first three print zones respectively. The bare PRINT statement, as in

```
10 PRINT "X","Y","Z"

20 PRINT
```

or

```
10 PRINT "X","Y","Z"\PRINT
```

can be used to separate the column headings from the numbers that are later printed under the headings. Two bare PRINT statements cause two lines to be skipped, etc.

For small amounts of output, quotes and variables can be combined:

```
70 PRINT "X=" X, "Y=" Y
```

would result in a printed line like this:

```
X= 4.67        Y=-4.32
```

assuming that the locations specified stored the indicated numbers. Note that the printer leaves a space for a sign but doesn't print anything there if the number is positive.

Semicolons are used to obtain close spacing. They cause the skipping of a single space (plus a second space if the number is positive) before printing begins:

```
70 PRINT "X="; X, "Y="; Y
```

would give just the same printed output as before. However,

```
70 PRINT X;Y;X
```

would give

```
4.67 -4.32  4.67
```

Now the payoff! You require only the command statements, and then you can begin remembering by practice what you've just read.

First you want to be able to completely clear the memory before you enter a new program. Typing something like **SCR** (for SCRatch) or **NEW** will accomplish this, depending on the particular installation. Or if your programs require a name, you may need to type something like **NEW PROB5** to begin business.

After you have typed in the program, you might want to see what the corrected version looks like. The **LIST** command or some variation of it accomplishes that. **LIST 30** then lists only line 30.

Each version of BASIC must have a method whereby the programmer can get access to the interpreter during execution, because it is all too easy to let the computer get caught in a loop or string of operations with no end. (Doesn't this remind you of the famous story of the *Sorcerer's Apprentice*?) You should learn this command with the same rapidity and enthusiasm that a new pilot demonstrates in learning the word MAYDAY! A typical command statement for this is **CTRL/C** (typing the **C** key while holding down the **CTRL** key).

To erase a single line of a program, type the statement number for that line followed by the RETURN key. To correct a single line, retype it with the same statement number.

The ultimate command statement, though, is the one that makes the magic fast-wit do your bidding, and that is the **RUN** command.

A tremendously useful feature of BASIC, resulting from its interactive nature, is that the computer's memory retains, and can be examined for, the last value for any variable *after* it has finished running a program (even if execution was interrupted with a command such as CTRL/C). Thus, after entering and running a trivial program with

```
10 T=4

20 S=T/4

30 T=T+1

40 END
```

we could type the immediate mode command **PRINT T** and get back an answer of 5. This memory examination capability is of *inestimable* value when seeking to determine the cause of an error message in a given line of a program or in removing "bugs" from a program that is coming up with strange answers. Suppose, for example, that the computer complains to you that you have illegally attempted to use zero as a divisor at line 45. You can counterattack by looking at that line, noting which variables are doing some dividing there, and quizzing the interpreter for their values to identify the guilty one.

Refer to Appendix I for a listing of typical keyboard commands and to Appendix III for typical error messages.

Review Questions

1. What are the largest and smallest numbers (in absolute value) that your BASIC can handle?
2. Why are the following variable names invalid? **3Z, XY, Y12, M***

3. How does the use of the **PRINT** statement in the program mode differ from its use in the immediate mode?
4. How would you obtain a printing of program lines from 10 to 70 inclusive?
5. What character in a **PRINT** statement causes the printer to move on to the next print zone?
6. Give two or three of the possible ways to instruct the printer to print **X= 7.42** right under **Y=−2.81** in the first print zone.
7. Devise a series of BASIC statements that calculates the quantity **Z** in problem 3 of the following and that uses only *one* pair of parenthesis marks.

Practice Programs: Write and run BASIC programs to make the indicated calculations, and print out the answer. Remember to clear the memory for each new program. Use the correction keys as needed when entering program lines. Be sure to make the output self-explanatory. Remember the command that allows you to stop execution at any time.

1. $h = v_0 t + \frac{1}{2}at^2$
   (uniformly accelerated motion)
   for $v_0 = 4$, $t = 2$, $a = 5$ — $h = 18$

2. $v = (v_0^2 + 2ax)^{1/2}$
   (uniformly accelerated motion)
   $v_0 = 4$, $a = 5$, $x = 20$ — $v = 14.6969$

3. $Z = \left\{R^2 + \left[\frac{1}{2\pi fC} - 2\pi fL\right]^2\right\}^{1/2}$
   (ac circuit impedance)
   $f = 60$, $R = 3.17$, $C = 38 \times 10^{-6}$, $L = .021$ — $Z = 61.9691$

4. $L = L_0 \left[1 - \left(\frac{v}{c}\right)^2\right]^{1/2}$
   (relativistic length contraction)
   $L_0 = 100$, $v = 1.71 \times 10^7$, $c = 3 \times 10^8$ — $L = 99.8374$

5. $N = N_0 \exp\left[-.693 \frac{t}{1.65 \times 10^{10}}\right]$
   (radioactive decay of thorium)
   $t = 1.65 \times 10^{10}$, $N_0 = 10.0$ — $N = 5.00$

E. The READ, DATA, and INPUT Statements

Much of the present-day importance of the computer is based on its ability to rapidly manipulate large quantitites of data. Data are commonly stored on cards, paper tape, magnetic tape, or magnetic disks. In this section it is shown that the keyboard is also useful in providing small amounts of data to the computer.

We have previously discussed only one way to assign a number to a variable—through the LET (or the implied LET) statement. Two other ways are through the combined READ-DATA statements (in which the data are contained within the program itself) and the INPUT statement (in which the data are provided by the programmer during program execution). The latter mode is favored when the programmer wishes (a) to have the data naturally appear with the output, or (b) to base some of his data on the results of a previous calculation (as in many types of numerical analysis or in games).

The general form of the READ statement is the word **READ** followed by variable names separated by commas. The DATA statement that goes with it is composed of the word **DATA** followed by a list of numerical constants separated by commas. When the READ statement is executed, the interpreter assigns the first number in the DATA statement to the first variable, etc. Thus the sample calculation in the last section could have been programmed as

```
10 READ A,T
20 S=A*T*T/2
30 PRINT S
40 DATA 20,5
```

If you provide more numbers than variables, the BASIC interpreter uses only as many as it needs. If you list more variables than numbers, though, an appropriate error message (such as ?00D = OUT OF DATA) is your reward.

The numbers in a DATA statement ordinarily can be used only *once*. But this can be overruled by inserting a

```
RESTORE
```

statement *before* the next READ statement. An example of this is the program

```
10 READ A,T
20 DATA 20,5,3
30 S=A*T*T/2
40 PRINT "S=" S
50 RESTORE
60 READ G,T,V0
70 X=V0*T-G*T*T/2
80 PRINT "X=" X
```

in which the following variable assignments are made:

```
A←20
T←5
G←20
T←5
V0←3
```

If there is more than one DATA statement, the interpreter uses them in the order of their statement numbers. RESTORE sends the interpreter back to the lowest-numbered statement in this case.

The DATA statement(s) can appear *anywhere* in the program, although it is considered good housekeeping practice to group them together at the beginning or end of the program.

The other assignment method is the **INPUT** statement, and it can be considered unique to terminal-based programming. It is composed of the word **INPUT** followed by a list of variables separated by commas. But now when program execution is begun, BASIC prints a question mark when it gets to the INPUT statement and expects the operator to type just as many numerical *constants* (separated by commas) as there are variables, followed as usual by the RETURN key. If the programmer types too many numbers, the excess is usually ignored; if he types too few, another question mark may be printed to request the remaining data, depending again on the particular BASIC interpreter.

Thus our overworked sample program could also have been written as

```
10 INPUT A,T
20 S=A*T*T/2
30 PRINT "S=" S
```

and then the output would look like this:

```
RUN
?20,5
S= 250
```

We can make the output more self-explanatory by combining a PRINT statement with the INPUT statement. For example, running the program

```
10 PRINT "A=";\ INPUT A
20 PRINT "T=";\ INPUT T
```

```
30 S=A*T*T/2
40 PRINT "S=" S
```

gives the attractive output

```
A=?20
T=?5
S= 250
```

where again the 20 and the 5 have been typed in by the programmer. Note how the use of the semicolon in the PRINT statement allows the question mark to be printed on the same line as the quoted characters.

Review Questions

1. What determines the location of the RESTORE statement?
2. Is the READ or the INPUT method the better choice when debugging a long program?
3. Write READ and DATA statements that assign the values 6.5 and 7.1×10^{-3} to variables A and $B0$ respectively.
4. Do the assignments indicated in question 3 above, but use the INPUT statement; make the output self-explanatory.

Practice Programs

1. $v = (v_0^2 + 2ax)^{1/2}$
   (uniformly accelerated motion)
   from READ statement:
   $v_0 = 4$, $a = 5$, $x = 20$
   $v = 14.6969$

2. $R = \dfrac{2 v_0^2 \sin \beta \cos \beta}{g}$
   (range for a theoretical projectile)
   $v_0 = 48$, $\beta = 23°$, $g = 32.2$
   $R = 51.4708$

3. $f = \dfrac{1}{2\pi}\left(\dfrac{1}{LC}\right)^{1/2}$
   (electrical oscillation frequency)
   $L = .01$, $C = 3 \times 10^{-6}$
   $f = 918.882$

4. $f = \dfrac{1}{2\pi}\left(\dfrac{k}{m}\right)^{1/2}$
   (spring oscillation frequency)
   from INPUT statement:
   $k = 7.1 \times 10^4$, $m = 36.2$
   $f = 7.04847$

5. $t = \dfrac{\tan^{-1}\,[V_0/(g/k)^{1/2}]}{(gk)^{1/2}}$
(time for object to fall)

$V_0 = 88$
$k = .001$
$g = 32.2$
$t = 2.54086$

6. $F = \dfrac{GM_1M_2}{R^2}$
(law of universal gravitation)

from READ statement:
$G = 6.67 \times 10^{-11}$
$M_1 = 1$
$M_2 = 500$

from INPUT statement:

$R = 5$	yields	$F =$	1.33400E–09
$R = 1$	yields	$F =$	3.33500E–08
$R = .037$	yields	$F =$	2.43608E–05

F. The GO TO, IF . . . THEN, STOP, and REMARK statements

Up to now we have had to run a program over again for each calculation involving a change in one of the variables. The simplest but least powerful way to improve on this is to use a GO TO statement to send the interpreter back to the beginning. For example, the program

```
10 A=32.2
20 PRINT "T=";\ INPUT T
30 S=A*T*T/2
40 PRINT "S=" S
50 PRINT
60 GO TO 20
```

would speed up the calculation of S for various values of T. (Only a *statement number* can be used after the GO TO in a GO TO statement.)

However, we really are seeking a method for automatically making many calculations by systematically varying one or more variables. This we can accomplish with the IF. . .THEN statement. What goes *between* the IF and the THEN is some sort of expression; if the expression is *true*, then the statement *after* the THEN is executed. If the expression is *false*, then control is passed to the *next statement*. For example

```
70 IF I=2 THEN J=2
```

would assign 2 to the variable **J** only if the condition (I=2) were true. Other valid uses of this statement follow.

```
70 IF I>=2 THEN GO TO 240
70 IF I<=2 THEN PRINT I
```

```
70 IF I<>2 THEN INPUT I
70 IF B*B-4*A*C<0 THEN STOP
70 IF SIN(X)<0 THEN X=-X
70 IF I=B+1 THEN IF J=1 THEN PRINT I
```

Study these examples closely. Note that any valid BASIC statement (including another IF statement) can follow the THEN. The expression "**IF I<=2 THEN. . .**" means "**IF I is equal or less than 2 THEN. . .**;" similarly, the expression "**IF I<> THEN. . .**" means "**IF I is not equal to 2 THEN. . .**"

The fourth example introduces the STOP statement. This statement is used to stop execution *before* the physical end (last line) of a program. The effect is equivalent to sending the interpreter to the END statement with a GO TO statement.

Suppose we wished to make the calculation $s = \frac{1}{2} at^2$ with t taking on values of 0 to 1 in increments of 0.01. The following program would do this:

```
10 PRINT \ PRINT "ACCEL", "TIME", "DISTANCE"
20 A=3 \ T=0
30 S=A*T*T/2
40 PRINT A,T, S
50 IF T>.99 THEN STOP
60 T=T+.01  GO TO 30
70 END
```

This is an appropriate time to point out once again that your relationship with the computer should be a master–slave relationship. You can do anything it can do (admittedly much slower), *plus* you can think of new things for it to do. A computer program can *always* be analyzed by taking the time to follow it step by step (often by inserting intermediate PRINT statements). Before you run a program that is supposed to do looping, you should check it out by going through the loop statements a couple of times to see if it is really going to make the desired calculation. Thus, for the above program

$A = 3$

$T = 0$

$S = 3 \times 0 \times 0/2 = 0$

$T = .1$

$S = 3 \times .1 \times .1/2 = .015$

$T = .2$

$S = 3 \times .2 \times .2/2 = .06$

. . .

and also at the end of the loop

$$T = .8$$
$$S = 3 \times .8 \times .8/2 = .96$$
$$T = .9$$
$$S = 3 \times .9 \times .9/2 = 1.215$$
$$T = 1.0$$
$$S = 3 \times 1 \times 1/2 = 1.5$$

Then you could make the computer print out each variable so that you can check it against your hand calculations.

Statement 50 in that last program was written as T>.99 rather than T=1 as an important insurance against round-off errors due to switching between different number systems. Because there is no *exact* binary equivalent for 0.01, T will never end up *exactly* equal to 1, so the equality would never be satisfied, and with most interpreters the looping would continue unchecked. (Try it!) What's more, this may be true even if the printing of the value of T includes the number 1, because the internal printing routine rounds off one place before printing! (This is a good time for you to recall your stop-execution command.)

Also, if the statement with line number 60 had been mistakenly written as

```
60 T=T+.01\ GO TO 20
```

(it has been done!), then the value of T would never exceed 0.01, and the looping also would not terminate without outside intervention.

Occasionally, someone writes a program containing a section that looks something like this:

```
50 IF X=4 THEN 60
60 Y=X+7
```

but it should be clear that nothing has been accomplished by the IF statement, because execution proceeds to statement 60 independently of the results of the comparison.

Caution must be exercised when using the IF. . .THEN statement in a multiple-statement line, because the behavior might *not* be equivalent to the same statements independently written. What happens is that the interpreter goes on to the next *line* as soon as a condition is *not* satisfied. In the execution of

```
40 R=5\ IF S<R THEN S=6  R=6
```

BASIC misses the last statement if S is equal to or greater than R. However, we can use this peculiarity to our own advantage in a calculation. Suppose we wish to calculate $S = \frac{1}{2}AT^2$ for the values $A = 32$, $T = 4$; $A = 19$, $T = 1.7$; $A = 8$, $T = 6$. Two good ways to make this calculation follow:

```
 10 PRINT\PRINT "ACCEL", "TIME", "DISTANCE"
 20 I=1
 30 A=32\T=4
 40 IF I=2 THEN A=19\T=1.7
 50 IF I=3 THEN A=8\T=6
 60 S=A*T*T/2
 70 PRINT A,T,S
 80 IF I=3 THEN GO TO 110
 90 I=I+1
100 GO TO 40
110 END
```

or

```
 10 PRINT\PRINT "ACCEL", "TIME", "DISTANCE"
 20 I=1
 30 READ A,T
 40 S=A*T*T/2
 50 PRINT A,T,S
 60 IF I=3 THEN GO TO 100
 70 I=I+1
 80 GO TO 30
 90 DATA 32,4,19,1.7,8,6
100 END
```

Note that we don't have to worry about number system round-off error when we use increments of 1, because 1 is an exact number in both base 10 and base 2 systems. Also, in the next section we see that looping often can be accomplished in a somewhat easier and more efficient fashion by using the FOR-NEXT statements.

Our programs are beginning to look more sophisticated, and it is time to consider the possibilities for "documentation." This 24 dollar word refers to leaving enough clues inside or outside a program so that you (and perhaps others) can in the future figure out what you

did and how you did it. Of prime assistance in this regard is the REMARK or REMark statement. For example, our last program could have begun with the statement

```
5 REM CALCULATION OF DISTANCE FROM ACCEL AND TIME
```

and the BASIC interpreter would store and list the statement but ignore it in execution. REMARK statements can be placed anywhere in a program, but you should not attempt to transfer to them with a GO TO statement because the interpreter may not keep track of their location during execution. Those who like to type may spell out the whole word or any other word beginning with REM.

Review Questions

1. What program statement would instruct the BASIC interpreter to print the value of a variable X only if X had a nonzero value?
2. Write a program statement that would skip to statement 70 if $\mathbf{X^2 - \sin(X)}$ is less than zero.
3. How could you instruct BASIC (in a single statement) to print X only if X is in the range from -25 to $+50$ inclusive?
4. How could a statement such as 40 IF I=2 THEN A=19\T=1.7 be written using only single statement lines?
5. How could you make sure that a certain loop was not executed more than ten times? (*Hint*: Use a summing variable inside the loop.)

Practice Programs: You should write BASIC programs to solve these problems *before* you get to the terminal—and check them over too. Only a representative sample of answers is given.

1. $T_C = \frac{5}{9}(T_F - 32)$ for T_F = 32 to 68 in increments of 4
   (temperature scale conversion)

T_F	T_C
32	0
36	2.22222
64	17.7778
68	20

2. $R = \dfrac{2V_0^2 \sin\beta \cos\beta}{g}$ for $V_0 = 9, g = 32.2,$ β = 20° to 70° in increments of 5°
   (range of a theoretical projectile)

β	R
20°	1.61695
25°	1.92701
65°	1.92701
70°	1.61695

3. $\theta = \tan^{-1}\left[\dfrac{V^2}{(Rg)}\right]$ for $R = 275, g = 32.2,$ V = 30 to 60 in increments of 5
   (angle of bank for aircraft)

V	θ (deg)
30	5.80347
35	7.87630
55	18.8609
60	22.1242

4. $P = 84.48877a^{1.5}$ for a = 1.02 to 1.04 in increments of 0.002

(satellite year)

a	P
1.02	87.0361
1.022	87.2922
1.038	89.3501
1.04	89.6084

G. The FOR-NEXT Statements

Because looping is so common, BASIC provides a second way that is more efficient and usually more readable and easier to use than the IF-THEN method. This is the FOR-NEXT loop, and it is the analogue of the FORTRAN DØ loop. In some versions of BASIC, this method accomplishes the incrementation about *five* times faster than the IF-THEN method, and therefore it should be used whenever a large number of loops is involved.

The loop is begun with the FOR. . . statement and closed with the NEXT. . . statement; both statements, using the same variable name, must always be present. As an example, the earlier program

```
10 PRINT\PRINT "ACCEL", "TIME", "DISTANCE"
20 A=3
30 T=0
40 S=A*T*T/2
50 PRINT A,T,S
60 IF T>.99 THEN GO TO 90
70 T=T+.01
80 GO TO 40
90 END
```

could have been written, equivalently,

```
10 PRINT\PRINT "ACCEL", "TIME", "DISTANCE"
20 A=3
30 FOR T=0 TO 1 STEP .01
40 S=A*T*T/2
50 PRINT A,T,S
60 NEXT T
70 END
```

During execution the interpreter first assigns the value 0 to the variable *T* and uses that value in all the statements up to the statement **NEXT T**. At this point it adds the increment (STEP .01) to the current value of **T** and compares this with the maximum (1). Because it is less than the maximum, it *then* assigns this new value to **T** and returns to the first statement after the FOR. . . statement and goes through the loop again. This automatic looping continues until the comparison value is greater than the maximum. When this happens, **T** usually will not be incremented, but in any case, execution passes to the statement immediately following **NEXT T**.

If desired, the loop can be left before the maximum is reached by use of the IF-THEN statement. Suppose that we wished to terminate the previous loop whenever **S** exceeded 2 or **T** exceeded 1, whichever occurred first. To do this we could simply add the statement

```
55 IF S>2 THEN GO TO 70
```

It is illogical, however, to transfer initially into the *middle* of a loop, because then BASIC has not been given the beginning, ending, or incremental values for the variable. With some BASIC interpreters, the result of such foolishness is to trap the machine inside the loop.

If the STEP value is 1, then that part of the statement can be omitted. Thus

```
40 FOR T=0 TO 5 STEP 1
```

is equivalent to

```
40 FOR T=0 TO 5
```

Variables to which numbers have been assigned previously can be used in place of numbers in the FOR statement. If we modified our program in this way:

```
35 INPUT T1, T2, D
40 FOR T=T1 TO T2 STEP D
```

we would have a great deal more flexibility during execution. Also, any valid BASIC expression can be used for the beginning, end, or step variables. Thus the statement

```
40 FOR X=LOG(Y) TO 4*LOG(Z) STEP Q*2*EXP(Z)
```

would be legitimate as long as Y, Z, and Q had been assigned numerical values previously. However, the 4*LOG(Z) and the Q*2*EXP(Z) would be evaluated *each* time through the loop, so in the interest of efficiency, *don't* use this form unless you really wish to be able to change the end point or the step value *in* the loop. Instead write this as

```
40 X1=4*LOG(2)
42 D=Q*2*EXP(Z)
44 FOR X=LOG(Y) TO X1 STEP D
```

Loops can be nested, that is, one enclosed within the other. Then the innermost loop is completely performed first, followed by an incrementation of the variable in the next outer loop, followed by another complete performance of the innermost loop, etc. For purposes of illustration, suppose that we wished to calculate $S = \frac{1}{2}AT^2$ (again!) for **T** taking on values from 1 to 2 in steps of 1 and for **A** taking on values from 6 to 10 in steps of 2. The following program would do this:

```
10 PRINT "T", "A", "S"
20 FOR T=1 TO 2
30 FOR A=6 TO 10 STEP 2
40 PRINT T,A,A*T*T/2
50 NEXT A
60 NEXT T
70 END
```

inner loop

outer loop

The output would look like this:

T	A	S
1	6	3
1	8	4
1	10	5
2	6	12
2	8	16
2	10	20

The number of nested loops usually is limited only by programmer courage and computer memory. But in every case an inner loop *must* be completely inside an outer one (try following through on paper the steps for a simple program in which this isn't the case, and you'll detect an undefined variable in there).

Because you now know how to make the computer work very long and hard, you should maintain efficient programming by making sure that assignment statements or calculations that do not change are located *outside* the loops. The program

```
10 FOR X=0 TO 10
20 A=1.4
```

```
30 B=4*A+6
40 Y=A+B*X
50 PRINT X,Y
60 NEXT X
```

is much less efficient than

```
10 A=1.4
20 B=4*A+6
30 FOR X=0 TO 10
40 Y=A+B*X
50 PRINT X,Y
60 NEXT X
```

In general the variable for the loop should *not* be altered in the loop. This sometimes happens rather innocently. Suppose we wish to calculate sin β for $\beta = 60°$ to $65°$. We might try a program like this:

```
10 FOR X=60 TO 65
20 X=X*PI/180
30 Y=SIN(X)
40 PRINT X,Y
50 NEXT X
```

Did you spot the bug? The problem is that in statement 30 the incrementing variable **X** has been changed from its initial value of 60 to a value of 60*PI/180 = 1.0472. This value of **X** is incremented by one at statement 50, but then it is again reduced when it returns to statement 20 on the second time through the loop. As a result **X** is growing smaller rather than larger, and the machine once again becomes ensnared in the infamous "infinite loop." A reasonable program that would have avoided this problem is:

```
10 FOR X=60 TO 65
20 Z=X*PI/180
30 Y=SIN(Z)
```

```
40 PRINT X,Y
50 NEXT X
```

A surprising number of beginning programmers have looped infinitely with yet another ingenious error. Sometimes you may want to skip some steps at the end of the loop under certain circumstances; when you do so, you should transfer execution to the NEXT statement and not to the FOR statement (which would reset the loop variable to its initial value).

Here is one more trap for the unwary programmer: If you accidentally specify a step value that has the wrong sign, as in

```
10 FOR I=1 TO 2 STEP -1
20 PRINT I
30 NEXT I
```

some interpreters won't execute *any* of the statements within the loop, and they won't give you an error message either! (The PRINT statement inside the loop would easily uncover the villain in this case.)

When the step value is a number less than 1, it does not have an exact binary equivalent when it is interpreted for execution. This leads to an accumulation of round-off error that could be significant. Thus the program

```
10 FOR I=-1 TO 0 STEP .01
20 PRINT I
30 NEXT I
```

prints the expected values for I until suddenly (on my computer) −.48001 instead of −.48 appears! Also, the last value of I is printed as −6.78003E−07 rather than the zero you reasonably might have expected. (Some versions of BASIC show this peculiarity with even fewer iterations.) If you think this might be a problem, you can take the route that some languages (as FORTRAN) always require you to take:

```
10 FOR I=-100 TO 0
20 J=I/100
30 PRINT J
40 NEXT I
```

Note that the general approach is to multiply the beginning and final variable values by the reciprocal of the original step value (to obtain a step value of 1) and then use a different variable to divide it back down in a succeeding statement.

If you leave a loop before it is completed (perhaps to do a calculation in a subprogram) but then return, the loop will resume with the current values for the variables.

This is as good a time as any to note that with multiple-statement lines you can write powerful miniprograms for execution in the immediate mode. For example, the previous program could have been written as

```
FOR X=60 TO 65\Z=X*PI/180\Y=SIN(Z)\ PRINT X,Y\NEXT X
```

In fact, any logical sequence of statements and commands that you can squeeze into a single line can be executed in the immediate mode.

Review Questions

1. Why must inner loops be completely contained within outer loops?
2. What immediate mode statement would cause the natural logarithms of numbers 10 through 50, step 10, to be calculated and printed?
3. What can cause a FOR-NEXT loop to be skipped over without execution?
4. Rewrite this statement to eliminate an accumulation of round-off error in the value of X.

```
FOR X=PI TO 7 STEP .4
```

5. Rewrite this statement to eliminate an accumulation of round-off error in the value of Y.

```
FOR Y=2.7 TO 9.1 STEP .02
```

Practice Programs: Carefully write out these programs before sitting down at the terminal, aiming for efficient execution and good-looking output. You may want to use more than one statement to reduce the number of required parentheses. Some representative answers are given.

1. $\frac{1}{f} = (n-1)\left(\frac{1}{R_1} + \frac{1}{R_2}\right)$ (lens formula)
   for $R_1 = 0.5, R_2 = 0.6$; $n = 1.5$ to 1.6 in increments of .02
   $n = 1.5$, $f = .545455$
   $n = 1.58$, $f = .470219$

2. $C_p = a + bt - \frac{c}{t^2}$ (heat capacity)
   for $a = 6, b = 7, c = 8$; $t = 100$ to 200 in increments of 20
   $t = 100$, $C_p = 705.999$
   $t = 180$, $C_p = 1265.97$

3. $V = \frac{4}{3}\pi r^3$ (volume of a sphere)
   for $r = 4$ to 9
   $r = 4$, $V = 268.083$
   $r = 8$, $V = 2144.66$

4. $P = \frac{nRT}{V}$ (ideal gas law)
   for $R = 8.317 \times 10^7$; $n = 4$; $T = 200$ to 400 in increments of 100; $V = 1$ to 3 in increments of 1
   $T = 200$, $V = 1$, $P = 6.65360E+10$
   $T = 400$, $V = 3$, $P = 4.43573E+10$

5. $m = \dfrac{m_0}{\left(1 - \dfrac{v^2}{c^2}\right)^{1/2}}$
(relativistic mass)

for $m_0 = 4, c = 3 \times 10^{10}$
$v = 2 \times 10^{10}$ to 2.9×10^{10} in increments of 0.1×10^{10}

v = 2.0E10, m = 5.36656
v = 2.9E10, m = 15.6226

6. $\lambda = \dfrac{2d \sin \theta}{n}$
(diffraction)

for $d = 1 \times 10^{-5}$
$n = 1.2$ to 1.6 in increments of 0.2
$\theta = 30°$ to 40° in increments of 5°

n = 1.2, θ = 30°, λ = 8.33333E–06
n = 1.6, θ = 35°, λ = 7.16971E–06

H. Subscripted Variables

You have previously had to write a separate variable name for each variable, which is particularly limiting when many variables are involved (as in large amounts of data). In normal mathematical practice, we commonly use a single variable with *subscripts* when the variables are related to one another. The same thing can be done in BASIC except that parentheses are used because subscripts aren't available.

Consider the problem of adding numbers and then obtaining their average. For five numbers

$$\text{Average} = \frac{(x_1 + x_2 + x_3 + x_4 + x_5)}{5}$$

Previously we would have made the calculation like this:

```
10 INPUT X1,X2,X3,X4,X5
20 S = (X1+X2+X3+X4+X5)/5
30 PRINT "AV=" S
```

With subscripted variables we can do it like this:

```
10 DIM X(5)
20 S=0
30 FOR I=1 TO 5
40 INPUT X(I)
50 S=S+X(I)
60 NEXT I
70 S=S/5
80 PRINT "AV=" S
```

Lines 30–60: These steps read the data and keep a running total.

Lines 70–80: These steps calculate the average and print it out.

The latter program appears to require a considerable amount of effort, but we can easily generalize it to find the average of a great assemblage of numbers:

```
10 DIM X(25 )
20 S=0
30 INPUT N
40 FOR I=1 TO N
50 INPUT X(I)
60 S=S+X(I)
70 NEXT I
80 S=S/N
90 PRINT "AV=" S
```

and now the program works for any number of variables between 1 and 250; the first input statement (line 30) tells BASIC how many numbers we are going to give it during execution.

Here are the general rules governing the use of subscripted variables.

1. At the beginning of the program, the BASIC interpreter should be told how much room (in its internal memory) to reserve for that variable. This is the function of the statement in line 10. **DIM** is the required abbreviation for **DIMENSION**. The subscript in the DIMension statement must be an integer constant (*not* a variable) between 1 and some large integer such as 32767.

   Although some BASIC interpreters assume a dimension of 10 when they encounter a subscripted variable that hasn't been dimensioned, use of this default option is to be discouraged as a sloppy and wasteful practice.

2. Any name that is valid for a nonsubscripted variable is also a valid name for a subscripted variable.

3. If you define a variable as being a subscripted variable but later in the program use that same name *without* a subscript (a very undesirable practice), the interpreter may or may not consider them to be distinct and separate variables. Don't do it.

4. Zero subscripts are allowed (unlike FORTRAN). But negative subscripts are strictly forbidden.

5. If you request the value of a subscripted variable for which you haven't yet assigned a value, some interpreters have been known to think about your request for about 12 seconds and then return with a humongous number like −.1702304E−9401!

6. If you use a noninteger number for the subscript, it will simply drop off the decimal part (this is called *truncation*). Thus X(1.9999) is interpreted as equivalent to X(1).

7. Any valid arithmetic expression can appear as the subscript, as well as an integer constant. The calculated value is subject to truncation as above. Thus

```
X(I+1)

X(2*I)

X(2*I+1)

X(2*I/J)

X(EXP(I))
```

are all valid expressions if (a) I and J are defined, and (b) the resulting subscript is within the allowed range.

You may and should be curious as to why 32767 is often the maximum allowable number for a statement line *and* for a subscript. This arises from the fact that many minicomputers use 16 binary digits to store the value of each integer variable. Fifteen of these are used to store the magnitude, and the remaining one is used to store the sign of the number. This implies that 2^{15} numbers can be stored, or numbers from 0 to 32,767. The term *binary digits* has been replaced by the contraction *bits*, and thus these minicomputers are said to have a *16-bit word*.

If you looked at the last program carefully, you noticed that it didn't really require the use of subscripted variables. It could have been written as

```
10 S=0

20 INPUT N

30 FOR I=1 TO N

40 INPUT X

50 S=S+X

60 NEXT I

70 S=S/N

80 PRINT "AV=" S
```

Notice too that the summing variable S has been initialized to zero in line 10 so that it has a value assigned to it for the first time through the loop when it needs it at line 50. This type of summing is very common.

In the interest of minimizing memory requirements, a general rule is that input data has to be stored only if it must be used at more than one point in the program. An example is the situation in which we wish to fit a curve to a set of data and then find the deviation of each point from this curve.

With dimensioned variables it is quite easy to ask for more internal memory than is available; each noninteger number requires *three* words of memory for storage. In addition the address (location) of each variable must be stored. With some BASIC interpreters, you can find out how much memory your program requires by typing a command such as **LENGTH** in the immediate mode. The response could be something like

```
62 WORDS USED, 2229 FREE
```

and this would tell you that you have used 62 of the 2291 words allotted to your terminal. If you made the request just after entering the program, it would be giving you the number of words required to store the program itself (plus one for the program name). It is not until execution is begun that memory is reserved for the variables. With a command like this, you can determine that a simple statement such as **10 A=1** requires four words for storage of the program instruction and an additional seven words for execution. (One more word would have been required if you had used the word **LET**.)

As another example of the use of dimensioned variables, suppose you have made a number of measurements of the time that it takes an object to fall a particular distance under circumstances for which air drag is negligible. (Perhaps you spent the afternoon dropping pennies into a wishing well and timing their fall?) You can determine a good value for this distance by using the average time in the appropriate distance equation, the familiar $S = \frac{1}{2}AT^2$. The following program would make this calculation for N observations (N = 1 to 40).

```
10 DIM T(40)
20 A=-32.2
30 INPUT N
40 FOR I=1 TO N
50 PRINT "T("; I ; ")=";
60 INPUT T(I)
70 NEXT I
80 T1=0
90 FOR I=1 TO N
100 T1=T1+T(I)
110 NEXT I
120 T1=T1/N
130 S=A*T1*T1/2
140 PRINT "CALCULATED DISTANCE=" S
```

These statements input N values for the time.

These statements calculate the average time.

Here the distance is calculated and printed.

(You might note again that the calculation really doesn't require subscripted variables or more than one loop.)

Statement 80 is used to make certain that the summing variable has a value of zero before it enters the loop, as before. Without this I have seen some computers come up with a different answer each time a program was run, so that should explain my nagging.

Line 50 illustrates a particularly useful way of making the printed output self-explanatory with a minimum of programmer effort. Remember that everything between quote marks is simply printed as it stands, but variables have their current numerical value printed. So the first time through the loop, when $I = 1$, the output looks like this:

```
T( 1 )=?
```

and BASIC expects to be given a value for $T(1)$. The 1 was printed because that is the current value assigned to I.

After being given a number by the programmer, the loop begins a second time with I now equal to 2:

```
T( 2 )=?
```

If the times were 1.41, 1.39, 1.43, and 1.42, then the terminal display would look like this:

```
T( 1 )=? 1.41
T( 2 )=? 1.39
T( 3 )=? 1.43
T( 4 )=? 1.42
CALCULATED DISTANCE= -32.1220
```

where the numbers after the question marks were typed by the programmer during execution.

Two or more subscripts are permitted in many versions of BASIC. As an example of their use, consider the general form of two simultaneous equations in two unknowns:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= c_1 \\ a_{21}x_1 + a_{22}x_2 &= c_2 \end{aligned}$$

as in, for example,

$$\begin{aligned} 4x_1 - 3x_2 &= 6 \\ -2x_1 + 4x_2 &= -5 \end{aligned}$$

One of the two solutions to the general equations is

$$x_1 = \frac{c_1 a_{22} - a_{12} c_2}{a_{11} a_{22} - a_{12} a_{21}}$$

and therefore a BASIC program to find only this solution could be written as

```
10 DIM A(2,2),C(2)
20 FOR I=1 TO 2
30 FOR J=1 TO 2
40 PRINT "A("; I ; ","; J ; ")=";
50 INPUT A(I,J)
60 NEXT J
70 NEXT I
80 PRINT "C(1), C(2)=";
90 INPUT C(1), C(2)
100 D=A(1,1)*A(2,2)-A(1,2)*A(2,1)
110 X1=C(1)*A(2,2)-C(2)*A(1,2)
120 X1=X1/D
130 PRINT "X1=" X1
```

These statements input the data.

These statements make the calculation.

When this program is executed for the example pair of equations, we obtain

```
RUN
A( 1, 1 )=? 4
A( 1, 2 )=? -3
A( 2, 1 )=? -2
A( 2, 2 )=? 4
C(1), C(2)=? 6,-5
X1= .9
```

where again the numbers after the question marks were typed during execution.

Review Questions

1. How many numbers can be stored under the variable **V** if it has been dimensioned by **DIM V(10)?**
2. How many words of internal memory are reserved by **DIM V(10)?**

3. What is the largest number that can be stored in an 8-bit word, assuming one bit is reserved for the sign?
4. If a subscript is given as **I = 2*J/3** and if **J = 1**, what is the effective value of the subscript?
5. Estimate the total memory requirement for the statement

```
10 J=J+1
```

6. Write two statements that allow you to enter values for **S3(I)** in a self-explanatory fashion, using an INPUT statement.

Practice Programs

1. Write a program to calculate the average and the average square value of a group of up to 15 numbers. Use subscripted variables and FOR-NEXT loops. Then check your program with the following data.

$x_1 = 5$ $\quad x_4 = -1$
$x_2 = -3$ $\quad x_5 = 4$
$x_3 = 2$

Answers: $\bar{x} \equiv \dfrac{\sum_{i=1}^{5} x_i}{5} = 1.4$; $\overline{x^2} \equiv \sum_{i=1}^{5} \dfrac{x_i^2}{5} = 11$

2. Add to the program above a calculation and printing of the deviation of each number from the average value.
3. $\dfrac{1}{R} + \dfrac{1}{R_1} + \dfrac{1}{R_2} + \dfrac{1}{R_3} + \dfrac{1}{R_4} + \dfrac{1}{R_5} + \dfrac{1}{R_6} + \dfrac{1}{R_7} + \dfrac{1}{R_8}$ (resistors in parallel)

Calculate R from the above formula, using subscripted variables and FOR-NEXT loops, for up to 15 numbers. Then check your program with these numbers: $R_1 = 6, R_2 = 8, R_3 = 4, R_4 = 10, R_5 = 7, R_6 = 3, R_7 = 5, R_8 = 9$.

Answer: $R = .699806$

4. $x_c = \dfrac{m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 + \ldots + m_nx_n}{m_1 + m_2 + m_3 + m_4 + \ldots + m_n}$ (one-dimensional center of mass)

Write a program to make this calculation for up to ten x's and m's; then run your program for

$m_1 = 7$ $\quad x_1 = -3$
$m_2 = 2$ $\quad x_2 = 5$
$m_3 = 4$ $\quad x_3 = 2$

Answer: $x_c = -.230769$

5. The Hermite polynomials are a group of polynomials that are solutions to the quantum mechanical oscillator problem. The first three of these are

$$H_0(X) = 1$$
$$H_1(X) = 2*X$$
$$H_2(X) = 4*X*X-2$$

The polynomials have the useful property that any one of them can be obtained from the two polynomials of next lower order through the relationship

$$H_N(X) = 2*X*H_{N-1}(X) - 2*(N-1)*H_{N-2}(X)$$

As an example, for $N = 2$,

$$H_2(X) = 2*X*H_1(X) - 2*(1)*H_0(X)$$

and substitution of $H_1(X)$ and $H_2(X)$ from above shows this to be correct.

Devise a program that will calculate $H_N(X)$ for any N starting with $H_0(X)$ and $H_1(X)$ as known polynomials in X. Then check your program by finding the value of $H_{10}(1)$ and $H_{10}(2)$. Answers: 8224, 200416

I. User-Defined Functions

When you execute a statement such as **Y=EXP(−4)**, the BASIC interpreter goes to its internal memory bank and finds that someone has deposited there a meaning for EXP in terms of a power series. It takes the numerical value of whatever is inside the parenthesis, plugs it into the series, and returns with the desired number.

In a similar fashion, the programmer can make up to 26 function DEFinitions of his own, and the interpreter will obediently look up the definitions and use them when they are needed later in the program. The general form of the defining statement is

DEF FNa(list of variables separated by commas) = BASIC expression

in which "a" stands for any capital letter of the alphabet (thereby giving 26 possible definitions). The "BASIC expression" to the right of the equals sign can include constants, the variable or variables appearing in the variable list, and even other defined functions.

Suppose we wish to calculate tan (x) for $x = 1$ to 2 in steps of 0.1 and also for $x = 3$ to 10 in integer increments. It could be accomplished using a defined function in this fashion:

```
10 DEF FNT(Z) = SIN(Z)/COS(Z)
20 FOR I=1 TO 2 STEP .1
30 PRINT I, FNT(I)
40 NEXT I
50 FOR J=3 TO 10
60 PRINT J, FNT(J)
70 NEXT J
80 END
```

Note that the variables used in the defining statement are *dummy* variables; that is, any valid variable names can be used, because they just tell the interpreter *where* to place the value it finds in parentheses when it evaluates the function. Thus the defining statement in line 10 could have been written as

```
10 DEF FNT(X) = SIN(X)/COS(X)
```

The foregoing example had only one dummy variable, but some versions of BASIC allow up to five or so. Some versions also permit implied variables. Both of these extensions are illustrated by the program below.

```
10 DIM A(10),T(10)
20 DEF FNV(T) = V0*T
30 DEF FNS(A,T) = FNV(T) + A*T*T/2
40 FOR I=1 TO 10
50 INPUT A(I),T(I)
60 FOR V0=0 TO 10
70 S = FNS(A(I),T(I))
80 PRINT V0, A(I), T(I), S
90 NEXT V0
100 NEXT I
```

V0 here is an implied variable because it appears in the expression on the right but not in the variable list on the left in the defining statement. During execution the current value for the variable is used by the interpreter. This is an excellent example of the greater freedom for the BASIC programmer compared to programmers in other languages; there are fewer seemingly arbitrary sequence rules in BASIC, and almost everything that is logical will be interpreted in the expected fashion during execution.

The user-defined function is a very powerful and useful feature for scientific and engineering programming because of the many formulas and equations that are used. In general it should be used if the same equation appears in more than one or two places in a given program. But its most powerful use is in permitting a complex program to be written for functions in general and then to be adapted to any desired function by simply changing the DEF statement. Examples of this use are the general root-solving routines, numerical integration, and curve-fitting programs, all of which are discussed in Chapter III.

BASIC generates an error message if you try to make more than one defined function with the same identifying letter. But even recursive (cannibalistic) use, as in **Y = FNX(4*FNX(X))**, is legal.

Review Questions

1. What is wrong with this statement?

```
30 FNA(X) = X* SIN(X)
```

2. Can you detect disadvantages and advantages of defined functions with respect to program debugging?

Practice Programs

1. Calculate $y = \tan(x) + \tan(x^2) + \tan(x^3)$ for $x = 1$ to 9 in increments of 2, using a single defined function for the tangent and varying the argument to obtain the second and third terms.

   Some answers: $x = 1, y = 4.67222; x = 9, y = -1.1117$

2. Calculate y = hyperbolic tangent of x $\equiv \tanh(x) \equiv \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$ for $x = 4, 5$, and 7, using a defined function for the hyperbolic tangent

   Some answers: $x = 4, y = .999329; x = 7, y = .999998$

3. Calculate the Celsius equivalent for Fahrenheit temperatures of 32 to 68 degrees in steps of 4 degrees, using a user-defined function for T_C: $T_C = (5/9)(T_F - 32)$. Some sample answers are given at the end of Sec. F.

4. Calculate $y = x - \dfrac{x^2}{2} + \dfrac{x^3}{3} - \dfrac{x^4}{4}$ for $x = \sin(1)$ and $x = \log_e(2)$ [power series approximation to $\log_e(1 + x)$ for $-1 < x \leqslant 1$]

   Answers: $x = \sin(1), y = .5607; x = \ln(2), y = .50622$

5. The surface area of a sphere is given by $4\pi r^2$, its volume is given by $(4/3)\pi r^3$, and its moment of inertia about an axis through the center is given by $(2/5)mr^2$, where m is its mass and r is its radius. Using three defined functions, calculate these quantities for $m = 1$ and $r = 2$ and also for $m = 3$ and $r = 3$.

   Answers: $A = 50.2655, V = 33.5103, I = 1.6; A = 113.097, V = 113.097, I = 10.8$

6. The resistance of a series circuit to the flow of alternating current can be calculated from

$$Z = [R^2 + (X_L - X_C)^2]^{1/2} \quad \text{where } X_L = 2\pi fL \quad \text{and} \quad X_C = \frac{1}{2\pi fC}$$

   Write defined functions for each of X_L, X_C, and Z (the last one using the first two). Make f an implied variable. Then, using subscripted variables, calculate Z for

$f = 60, R = 20, L = 15 \times 10^{-3}, C = 7 \times 10^{-6}$	Answer: 373.821
$f = 120, R = 300, L = 1 \times 10^{-2}, C = 1 \times 10^{-5}$	Answer: 325.034
$f = 1000, R = 7, L = 1 \times 10^{-3}, C = 5 \times 10^{-4}$	Answer: 9.19672

J. Subroutines

Subroutines provide a way to leave a program, do a set of calculations or print some numbers, and then return to the calling program. The initiating command is the phrase **GO SUB**

followed by a statement line number. The statement that returns execution to the main (calling) program is (surprise!) **RETURN**. The return is to the statement immediately following the GO SUB statement.

The advantages of using subroutines include:

1. If a certain calculation has to be done more than once in a program, it saves having to type the calculation more than once. This is somewhat similar to the advantage of using defined functions except that any number of output variables can be obtained, rather than just one, and printing can also be done in the subroutine.
2. Program memory requirements can be reduced.
3. Calculations that are made in more than one program need to be written and debugged only once. A very useful program of this type is one that solves in general a set of simultaneous equations (Chap. III).

To illustrate, suppose we wish to make the distance calculation of Sec. H for both average time and average acceleration. We could do this with a single subroutine (lines 1000 to 1050) to calculate the averages, as follows.

```
10 DIM X(40)
20 PRINT "NO. OF TIME OBSERVATIONS=";
25 INPUT N
30 FOR I=1 TO N
40 PRINT "T("; I; ")=";
45 INPUT X(I)
50 NEXT I
60 GO SUB 1000
70 T1=X1
80 PRINT "NO. OF ACCELERATIONS OBSERVATIONS";
85 INPUT N
90 FOR I=1 TO N
100 PRINT "A("; I; ")=";
105 INPUT X(I)
110 NEXT I
120 GO SUB 1000
130 A1=X1
```

The time observations are given to the interpreter here.

The average time is calculated here.

The acceleration observations are given to the interpreter here.

The average acceleration is calculated here.

```
140 S=A1*T1*T1/2
150 PRINT "CALCULATED DISTANCE=" S
160 STOP
1000 X1=0
1010 FOR I=1 TO N
1020 X1=X1+X(I)
1030 NEXT I
1040 X1=X1/N
1050 RETURN
1060 END
```

The final calculations and printing are accomplished here.

The subroutine is typed immediately following the main program.

Note that we used the same variables for both time and acceleration, because those are the variables used in the subroutine. If we had needed to save the input variables for later calculations, we could have used different variable names but transferred the values to X(I) and N just before going to the subroutine.

When execution arrives at statement 60, it transfers to statement 300, continues there until it reaches the RETURN statement, and then automatically transfers back to statement line 70. The subroutine calculates the average as X1, so statement 70 saves that value as variable T1. Statements 80 through 130 similarly calculate the average acceleration, and statements 140 and 150 make the final distance calculation and printing.

On statement line 160, the STOP statement that was introduced in Sec. F appears again. Here it tells the computer that it is through with the calculations even though there are statements beyond it. (The STOP statement can be usefully employed in debugging, too; it can stop a program in the middle so that you can determine the current values of variables.) Without this STOP statement, the execution would continue into the subroutine without a GO SUB statement, which we don't want. (The interpreter would rebuke us with something like "RETURN WITHOUT GO SUB" and stop anyway.)

Subroutines should be grouped at the end of the main program, in general. A subroutine that you expect to use with other programs should use very large line numbers to avoid cramping the available room above it.

Subroutines can be nested; that is, one subroutine can call another. Each time a GO SUB statement is encountered, BASIC marks its location so that it can return to execute the following statement as soon as it encounters a RETURN statement. A RETURN statement without a GO SUB statement (and vice versa) is clearly illogical, but almost anything else is fine, because BASIC executes on a line-by-line basis rather than by trying to make an overall translation first. Thus multiple RETURN statements and multiple entry points into a single subroutine are quite legal and useful.

Review Questions

1. What can a subroutine do that a defined function can't do?
2. Is statement line 1000 in the example really necessary, assuming the interpreter initializes all variables to zero when beginning execution?
3. What general guidelines do you think should be followed when inserting IF statements or GO TO statements into subroutines?

Practice Programs

1. Write a subroutine to calculate $y = x_1 + 2\,x_2 + x_3$. Provide the three variables, and print the answer in a main program, for

   $x_1 = 1, x_2 = 4, x_3 = 2$ Answer: 15

   $x_1 = 3, x_2 = 1, x_3 = 7$ Answer: 26

2. Write a subroutine to calculate the average of the *sum* of the squares of a set of numbers (i.e., $\Sigma\, x^2/N$). Use subscripted variables and allow for up to ten data points. Give the input variables to the interpreter in the main program, but do all the calculations in the subroutine. Then check your program for four data points:

   $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$ Answer: $\displaystyle\sum_{i=1}^{4} \frac{x_i^2}{4} = \frac{1^2 + 2^2 + 3^2 + 4^2}{4} = 7.5$

3. Write a subroutine to calculate the area and volume of a sphere of radius R, where $A = 4\pi R^2$ and $V = (4/3)\pi R^3$. Then write a main program to calculate and print A and V for $R = 1$ to 9 in steps of 2. (The GO SUB statement can be placed inside a loop for this.)

   Some answers: $R = 1; A = 12.5664, V = 4.18879$
   $R = 3; A = 113.097, V = 113.097$
   $R = 9; A = 1017.88, V = 3053.63$

4. Devise a subroutine that calculates the tangent of X if a variable N is equal to 1, the secant of X [$=1/\cos(X)$] if $N = 2$, and the cosecant of X [$=1/\sin(X)$] if $N = 3$. Then write a main program that obtains values for N and X through an INPUT statement, goes to the subroutine for the calculation, and returns to the main program to print the result.
5. Devise a subroutine that calculates and prints any of the Hermite polynomials (refer to problem 5 at the end of Sec. H). Place $H_0(X)$ and $H_1(X)$ in the subroutine, and calculate higher orders through the equation that relates them to lower orders. Write a main program that obtains the order of the polynomial and the value for X in an INPUT statement and returns the proper value. Test with some of the known values.

K. Additional Output Control: TAB and CHR$ Functions

The TAB function gives you greater flexibility and control over the printing (or video display) of program output. A statement such as

```
20 PRINT TAB(20); "SPEED"
```

moves the printer to column 20, and the semicolon causes it to begin printing immediately thereafter.

TAB(X) is a particularly useful function when you have more than five columns of output. If the full twelve spaces of a print zone are not needed for a certain column, then the next column can be moved in closer to it. For example,

```
20 PRINT "T="; TAB(8); "V(FT /SEC)"

. . .

70 PRINT T; TAB(8); V
```

prints the second column close to the first for both the column heading and the variable value.

The argument for the TAB function can be a variable as well as a constant. When the variable is evaluated, any fractional part is discarded; therefore, TAB(21.8) is interpreted in the same fashion as TAB(21).

CHR$(X) is another useful function for output control. It prints all the alphabet (X = 65 to X = 90), all the numbers (X = 48 to 57), and all the other characters on the keyboard (X = 33 to 47, 58 to 64, and 91 to 96). In addition it rings the terminal bell (X = 7), causes a line feed (X = 10), or causes a carriage return (X = 13). X in this case is the integer assigned to the character or function in the American Standard Code for Information Interchange (ASCII for short). As an example,

```
10 PRINT CHR$(7); CHR$(33)
```

rings the bell and prints ! when executed.

Lest you immediately conclude that bell-ringing is only for game players, let me assure you that it can and has had serious business to do. It can be placed in a program to tell the programmer when it gets to that point, how long it takes to do a loop including that statement, etc.

The argument for the CHR$(X) function can be a variable and any fractional part is truncated, as for the TAB(X) function.

Practice Programs

1. Use the TAB function to print the following in the columns indicated.

Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Character (a)	T	I	M	E				A	C	C	E	L			I	N	I	T	V			D	I	S	T
(b)	1	.	1	5			3	2	.	2	2	2			4	2	.	6	1			1	3	7	3
(c)	R				C					L					F								"	Z	"
(d)		H	O	(	X	)							H	2	(	X	)				H	3	(	X	)

2. Combining the TAB and CHR$ functions (with variable arguments) in a loop, print the letters of the alphabet in a slanted line down the page.

3. Devise a page divider by printing five asterisks, separated from each other by two spaces, in the middle of the page.

4. Print a square using the ↑ for the top and bottom and the > for the sides. Have the computer ring its bell when it has finished.

L. Storing Programs and Data: External Memory

Big computer programs from little ones grow. If a small amount of data is good, then a large amount is even better. For these reasons programmers quickly searched for practical ways to store programs and data. Traditionally magnetic tape and hard magnetic disks have been used, but in low-cost systems, paper tape, cassette tape (based on the audio cassette pioneered by Philips), and flexible (floppy) magnetic disks are quite popular. The information on a disk can be accessed much more quickly than on a tape (random access versus sequential access), but there is an accompanying cost penalty.

The complexity of the BASIC commands to store programs and data is directly proportional to system complexity, but the usual command statements for programs are **SAVE** and **OLD**. If the system has more than one mass storage device, the particular device must be specified. Thus, in one system, the command

```
SAVE DX1:
```

stores a program on the floppy disk that is now in disk drive number 1. If the name of this program happened to be **RK2**, it could be returned to the internal memory with the command

```
OLD DX1:RK2
```

For our purposes it is desirable that you be able to store your programs and data, because the following chapters require longer programs that can't be entered and debugged in a single session at the terminal, and some of your programs will be of general use for later problems.

M. Extensions to BASIC

All of the basic elements of BASIC have now been presented. If you are able to use these intelligently, you are ready for some very meaningful problems and in particular for the succeeding chapters of this book. Yet many Extended BASIC interpreters are available, and an awareness of any additional features possessed by your system can enhance your programming efficiency. This section presents some common extensions to BASIC that are applicable to programming for the physical sciences.

Computed **GO TO** statements and computed **GO SUB** statements are sometimes useful. The statement

```
50 ON N+1 GO TO 70,80,90
```

sends the interpreter to statement lines 70, 80, or 90 based on whether the expression (N+1) is equal to 1, 2, or 3 respectively. Other values for the expression transfer execution to the next statement. Similarly, the statement

```
50 ON 4*X-Y GO SUB 70,90,100
```

transfers execution to a subroutine beginning at lines 70, 90, or 100 if the expression (4*X–Y) has a truncated value equal to 1, 2, or 3 respectively.

Some systems allow the programmer to treat an array of numbers specified by a subscripted variable as a *matrix*. If so, special BASIC statements generate matrix operations such as addition, multiplication, and inversion. With this capability, systems of equations are manipulated rather easily; an example is the solution of simultaneous equations. But because you may not be familiar with matrix algebra and because many systems don't implement this feature, these statements are not used in this text.

One implementation of string variables and precise format control is now described. A string of characters that has no intrinsic meaning to the computer is known as a "string expression" or a "string." Thus, in the BASIC statement

```
10 PRINT "COMPUTERS ARE FUN"
```

the characters between quotes constitute a string constant. Extensions of BASIC permit the use of string *variables* that bear the same relationship to string constants as numeric variables do to numeric constants. This capability leads to precise control of the output format that can be quite useful for complex programs.

Any valid numeric variable name when terminated with **$** becomes a valid string variable name. Subscripted string variables also are permitted.

String variables are assigned values in the same fashion as numeric variables:

```
LET A$="ENTERPRISE"

A$="ENTERPRISE"

READ A$ followed by DATA "ENTERPRISE"

INPUT A$ answered by ENTERPRISE
```

all have an equivalent effect. Note that quotes are always used in assigning values to string variables except when the INPUT statement is used–but they are never construed to be part of the variable. A string variable can be nulled with a statement such as

```
A$=""
```

because then there is nothing (not even a space) between the quote marks.

Following is a sample program using string variables, with output from it.

```
10 READ A$, B$
20 C$="!"
30 PRINT "MY NAME IS " B$; A$; C$
40 DATA "DUCK", "DONALD"
RUN
MY NAME IS DONALD DUCK!
```

Detailed control of printed output is accomplished through use of a string expression as a coded "image" of the output (much as "PICture" does in the COBOL language). The string expression has special characters to indicate where the output variables are to be inserted, their relative location ("justification") if they are strings, and the location of the decimal point if they are numbers.

The following statement prints the results of a calculation of distance travelled, rounded off to the nearest tenth:

```
100 PRINT USING "DISTANCE TRAVELED IS +,###.# FEET.", S
```

If the distance **S** has a calculated value of 421.364, then the generated output line is

```
DISTANCE TRAVELED IS  +421.4 FEET.
```

The string expression enclosed in quotes immediately after PRINT USING is the image or format for the output; it is always followed by a comma and then the list of variables or constants to be printed within that format. The + character signals the beginning of a numeric field format and reserves a single position for a + or – sign. The # characters are *place holders*; they reserve positions for numbers after the sign. The . character within the numeric field specifies the location of the decimal point (between the last and next-to-last number); outside the numeric field, as at the end, it is treated as any other character. The , character also reserves a position but is printed only if it can have numbers on both sides of it. If **S** equals 1421.364, then the generated output is

```
DISTANCE TRAVELED IS +1,421.4 FEET.
```

On the other hand, if **S** equals 71421.36, then the generated output is

```
DISTANCE TRAVELED IS *******
```

The asterisks may conceal some dark thoughts the computer has about you, but for certain they do constitute a necessary and sufficient warning that the number is too large for the allotted space (FORTRAN does this to the programmer all the time!).

In the absence of a decimal point within a numeric field, the number is *rounded* to its integer value when printed. Note that in any case the printed number is *rounded off* rather than truncated.

Equivalently, the format string expression can be split off from the list of variables by using a string variable for it:

```
100 A$="DISTANCE TRAVELED IS +,###.# FEET."
110 PRINT USING A$, S
```

and this is often more convenient.

A second and often preferred way to start a numeric field is with the − character; then a negative sign is printed for a negative number but a blank (available for use by a number) is left for a positive number.

It is also possible to specify the exponential format by entering a group of five consecutive up-arrows into the numeric field. Thus printing 14.781 with the format "−#.#↑↑↑↑↑" yields

```
1.5 E+1
```

and with the format "−###.#↑↑↑↑↑" the output is

```
147.8 E-1
```

Some additional examples follow.

FORMAT Field Specification	Number Stored	Number Printed
"−##"	143.6	144
"−##"	−143.6	***
"−##"	− 17.9	−18
"−##"	4.26	4
"+##."	− 7.836	− 8.
"+##."	48.82	+49.
"+##."	746.3	****

Many variables can be printed in one PRINT USING statement. Consider the following example.

```
10 R=150
20 C=1.2E-6
30 L=1.5E-3
40 A$="R=-### OHMS, C=-#.#↑↑↑↑↑ FARADS, L=-#.#### HENRYS."
50 PRINT USING A$, R,C,L
RUN
R=150 OHMS, C= 1.20 E-6 FARADS, L= 0.0015 HENRYS.
```

If you supply more variables than there are format fields, then a carriage return and line feed are generated and the format expression is reused. If you supply more fields than there are variables, on the other hand, printing stops at the first unneeded field.

You can generate close printing of succeeding output by using a semicolon (but not a comma) after the last variable in the list.

Here is a summary listing of the special characters used in numeric format fields.

Character(s)	*Function*
+	Starts field; prints plus or minus sign.
–	Starts field; prints minus sign if negative.
↑↑↑↑	Specifies exponent format.
.	Specifies location of decimal point and prints decimal point.
#	Place holder.

Review Questions

1. Assuming an internally stored number of 1783.46, what numeric field specification prints each of the following?
   (a) +1783.46
   (b) 17.83 E+2
   (c) 1.783 E+3
   (d) 1783
2. Write a single computed GO TO statement that sends execution to statement lines 50, 80, 110, and 140 if $X = -5, 0, +5$, or +10, respectively.
3. Write a single computed GO SUB statement that sends execution to statement lines 100, 200, and 300 if $X < Y$, $X = Y$, or $X > Y$, respectively. (*Hint*: You haven't forgotten the SGN function already, have you?)

Practice Programs

1. Check out your answers for the questions above.
2. Input a speed in miles per hour, and calculate the equivalent in feet per second by multiplying it by 5280/3600. Try different values for the input speed, and print the results in the format

```
60 MPH EQUALS 88.0 FT/SEC.
```

3. Use a PRINT USING statement to print the following headings; then use another PRINT USING statement to print out the numbers using the variables $E1 = 43$, $T = 4.3001$, and $F = 59.999$.

EXP	TIME	FREQ
43	4.3 SEC	60.0

II

Flowcharting

A. Rationale, Symbolism, and Examples

Striding out to your aging but still very sleek twin-engined aircraft, you have real confidence that this time the flight is going to be a success--because this time you have loaded passengers and cargo with the assistance of your programmable hand calculator. To get this far in the world, and in programming, you better now learn about flowcharts.

Flowcharts serve two primary functions: (a) They provide documentation allowing others to understand how and what your program accomplishes; and (b) they permit the logic of a complex program to be checked *before* the program itself is written. (It turns out that everyone likes to use flowcharts, but unfortunately few seem to like to draw them. I hope to convince you to do it anyway.)

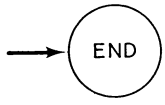
The first task for you is to study Fig. II-1 to become acquainted with the symbols I'll be using. Then as a simple example of a flowchart consider the calculation of f from the formula $1/f = (n - 1)\,(1/R_1 + 1/R_2)$, where $R_1 = .5$, $R_2 = .6$, and $N = 1.5$ to 1.6 in 0.02 increments. This is charted in Fig. II-2 and illustrates the usefulness of the looping symbol.

As an example of the second function of a flowchart--figuring out the logic--consider the calculation of the quantity $\log_e (1 + X)$ for $|X| < 1$ through its power series expansion:

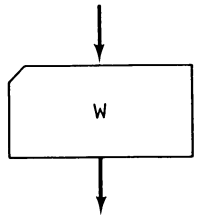
$$\log_e (1 + X) = X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \frac{X^5}{5} \cdots$$

Why don't we just make the calculation by writing LOG(1 + X) and be done with it? One explanation is that use of this power series could be much more efficient (converge more rapidly) than the built-in logarithm power series *if* the value of X is quite small.

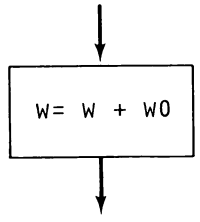
It appears that we should save each power of X as we calculate it so that the following term can be calculated from it by simply multiplying by X and changing the denominator. Also, we must have a criterion for ending the series when sufficient accuracy has been attained. "Sufficient accuracy" might be defined as the point where the last term added is less than 1×10^{-6}.



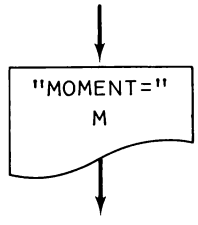
The circular symbol is used to indicate the beginning (START), execution end (STOP), or physical end (END) of a program. It is also used to show the connection between separated parts of a program or subprograms.



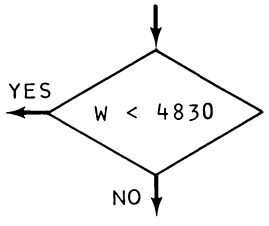
This symbol indicates the input of data to the program. (The shape is that of the common computer card.)



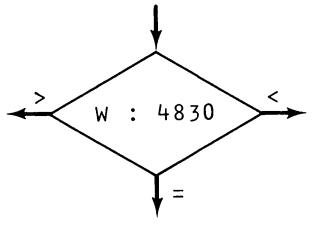
Calculations or assignment statements are placed within a rectangular box.



This symbol (a torn off sheet) indicates the output of identifying headings and variable values to the display or the printer.

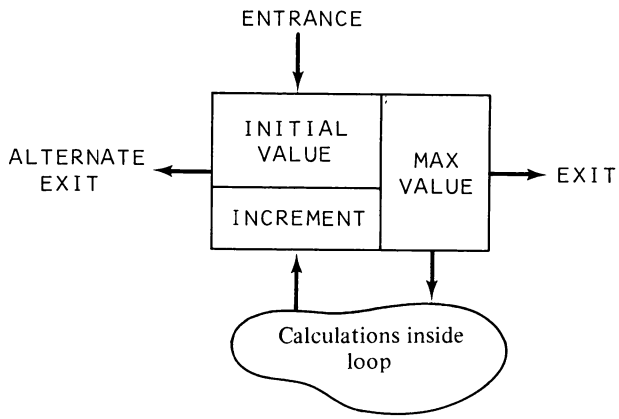


A diamond symbol indicates the logical comparison of a variable and a number or a variable and another variable.



This indicates another logical comparison but with a three-branch exit.

Figure II-1. Flowchart symbols



This is the least standard of the symbols, but it has great usefulness in indicating automatic looping.

Figure II-1. (Continued)

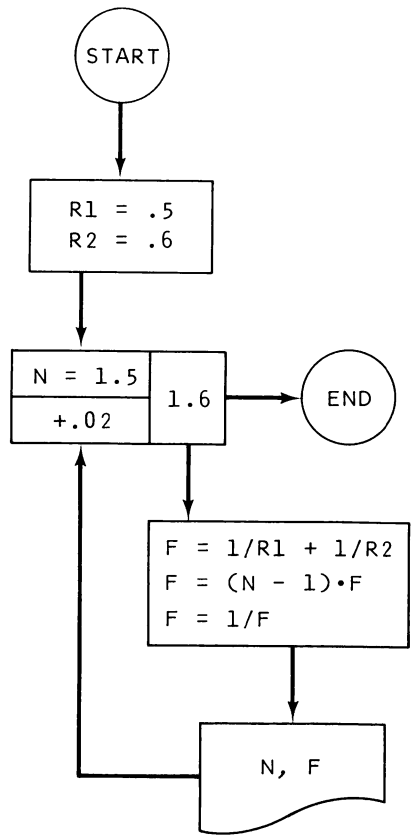


Figure II-2.

So if we use the variable A to save the powers of X, we can make the calculation in this way:

1. Assign $A = X, N = 1, Y = X$
2. Assign $A = -AX, N = N + 1, Y = Y + \frac{A}{N}$

3. If $\frac{A}{N} < 1 \times 10^{-6}$, then go to step 5
4. Go to step 2
5. Print X, Y, N
6. Stop

We check out our logic by looking at the steps we've written down and going through them a few times (look away and try it yourself first):

$A = X$
$N = 1$
$Y = X$
$A = -X^2$
$N = 2$
$Y = X + -\frac{X^2}{2}$
$A = X^3$
$N = 3$
$Y = X - \frac{X^2}{2} + \frac{X^3}{3}$

Figure II-3 gives the flowchart for this calculation and illustrates the "IF" symbol. (Could you have written out the program without these preliminaries?)

Now you should be ready to solve the problem of properly loading that aircraft. Basically we must make sure that the center of gravity (cg) is located in the proper region relative to the wing so that the plane can be maneuvered and landed safely. This is accomplished by first determining the moment contributed by the fuel, each passenger, and the baggage, and then checking the operating manual to see if the total weight and moment are acceptable. In detail the steps are:

1. Start with the empty weight and moment, 3170 lb and 106,200 in.-lb, respectively.
2. Add the weights of the pilot and front passenger and add that to the aircraft weight; multiply their total weights by 37.06 inches to find their contribution to the moment, and add that to the aircraft's empty moment.
3. Add the weights of rear-seat passengers; add their moments determined by multiplying their weights by 70.98 inches.
4. Add the weights of the individual items of baggage; determine that this total doesn't exceed 200 lb. Then add this weight to the total weight and add the moment determined by multiplying the total baggage weight by 96.11 inches.
5. Determine if the landing weight cg location is acceptable by checking to see if the moment falls between the maximum and minimum given by these formulas:

$$M_{\min} = (31.8)W + 1350 \qquad W \leqslant 3900$$

$$= (58.3)W - 102000 \qquad 3900 < W \leqslant 4830$$

$$M_{\max} = (41.38)W + 2140 \qquad W \leqslant 4830$$

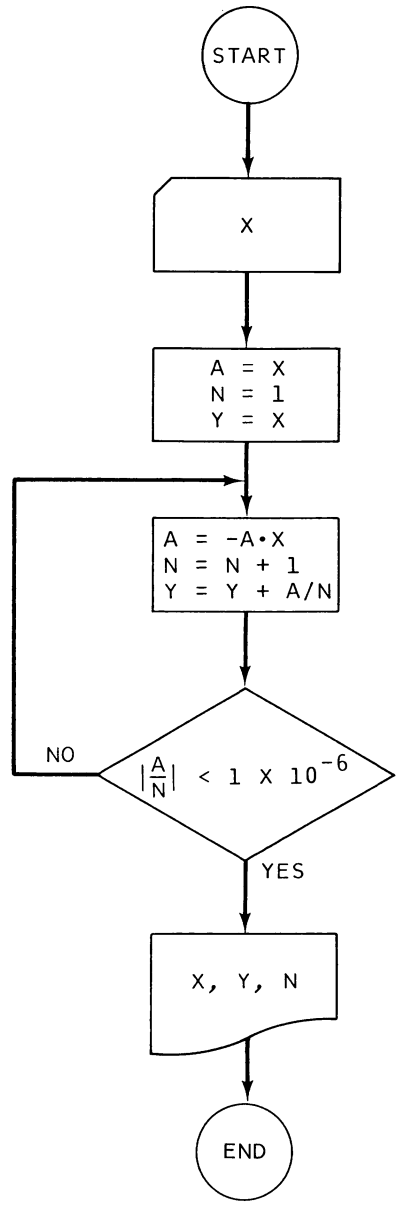


Figure II-3. Calculation of $Y = \text{Log}_e(1 + X)$

6. Add fuel at the rate of 6 lb per gallon until *either* the tanks are full (100 gallons and 600 lb) *or* the total aircraft weight has reached 4830 lb; add this weight and add the moment determined by multiplying the fuel weight by 35.00 inches. Print out a message indicating how much fuel can be put in the aircraft.
7. Determine if the take-off cg location is acceptable from the formulas given in step (5). Print a GO–NO GO message.

To get even this far we've had to organize the steps and the logic, and probably you could write the program right now. But I'll try to show you that drawing a flowchart requires you to bring the logic even closer to programming logic, and this is a necessity in a really complex program.

Perhaps the next step should be the assignment of variable names:

W = weight of aircraft (running total)

M = moment of aircraft (running total)

P0 = pilot's weight

P1 = front passenger's weight

P2 = back passengers' weights

N = number of items of baggage

W0 = weight of an item of baggage

W1 = running total of baggage weight

F = weight of fuel

Noting that we have to use the moment formulas of step (5) two times, it seems reasonable to put that calculation into a subroutine. Then after a couple of false starts, we come up with Fig. II-4 for the main program and Fig. II-5 for the subroutine. Study the charts carefully and convince yourself that they really do all the steps outlined earlier. (The point at which a step is begun is indicated by numbers in parentheses on the chart.) It is quite easy to use some sample data with a pocket calculator to make a check of the logic in the chart.

Do you see that now we have reached a point where a programmer could write a successful program without knowing anything about the problem? To be able to reach this point has become a fundamental skill for every scientist and engineer, whether or not he then writes the actual program.

Well go ahead and write out a program! For debugging purposes you'll want to print out the moment and weight each time you enter the subroutine. Here are some realistic data and my answers.

1. $P0 = 170$, $P1 = 100$, $P2 = 400$, $N = 1$, $W0 = 100$
   Answers: $W_{\text{landing}} = 3940$ lb, $M_{\text{landing}} = 154209$ in.-lb
   $W_{\text{take-off}} = 4540$ lb, $M_{\text{take-off}} = 175209$ in.-lb
   Fuel = 100 gal "GO FLY!"
2. $P0 = 200$, $P1 = 170$, $P2 = 600$, $N = 5$, $W0 = 30, 100, 12, 8, 50$
   Answers: $W_{\text{landing}} = 4340$ lb, $M_{\text{landing}} = 181722$ in.-lb
   $W_{\text{take-off}} = 4830$ lb, $M_{\text{take-off}} = 198872$ in.-lb
   Fuel = 81.6667 gal "GO FLY!"
3. $P0 = 90$, $P1 = 0$, $N = 0$
   Answers: $W_{\text{landing}} = 3260$ lb, $M_{\text{landing}} = 109535$ in.-lb
   $W_{\text{take-off}} = 3860$ lb, $M_{\text{take-off}} = 130535$ in.-lb
   Fuel = 100 gal "GO FLY!"
4. $P0 = 170$, $P1 = 150$, $P2 = 800$, $N = 2$, $W0 = 125, 70$
   Answers: $W_{\text{landing}} = 4485$ lb, $M_{\text{landing}} = 193585$ in.-lb
   "M IS TOO LARGE"

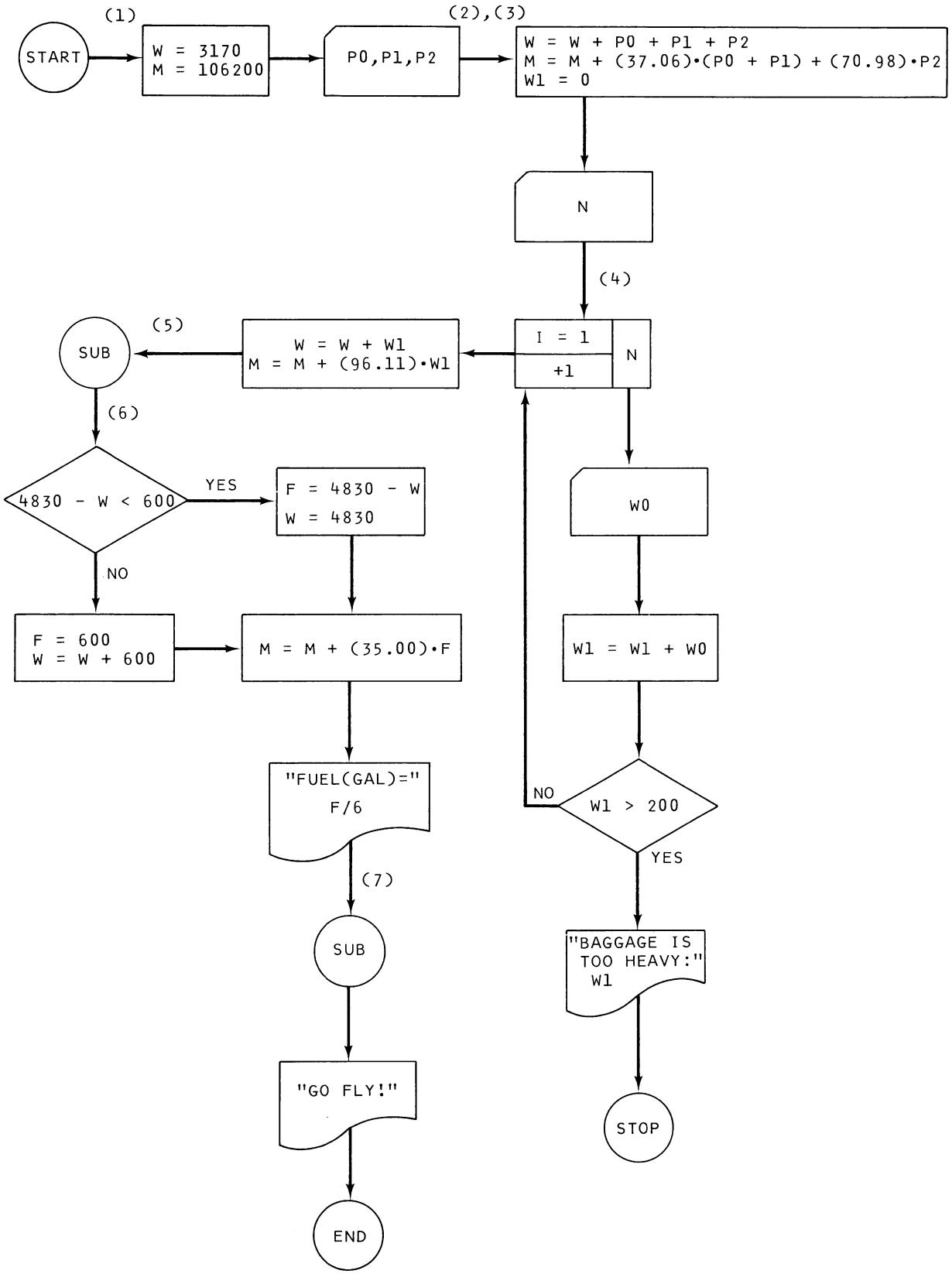


Figure II-4. Main program for the aircraft loading problem

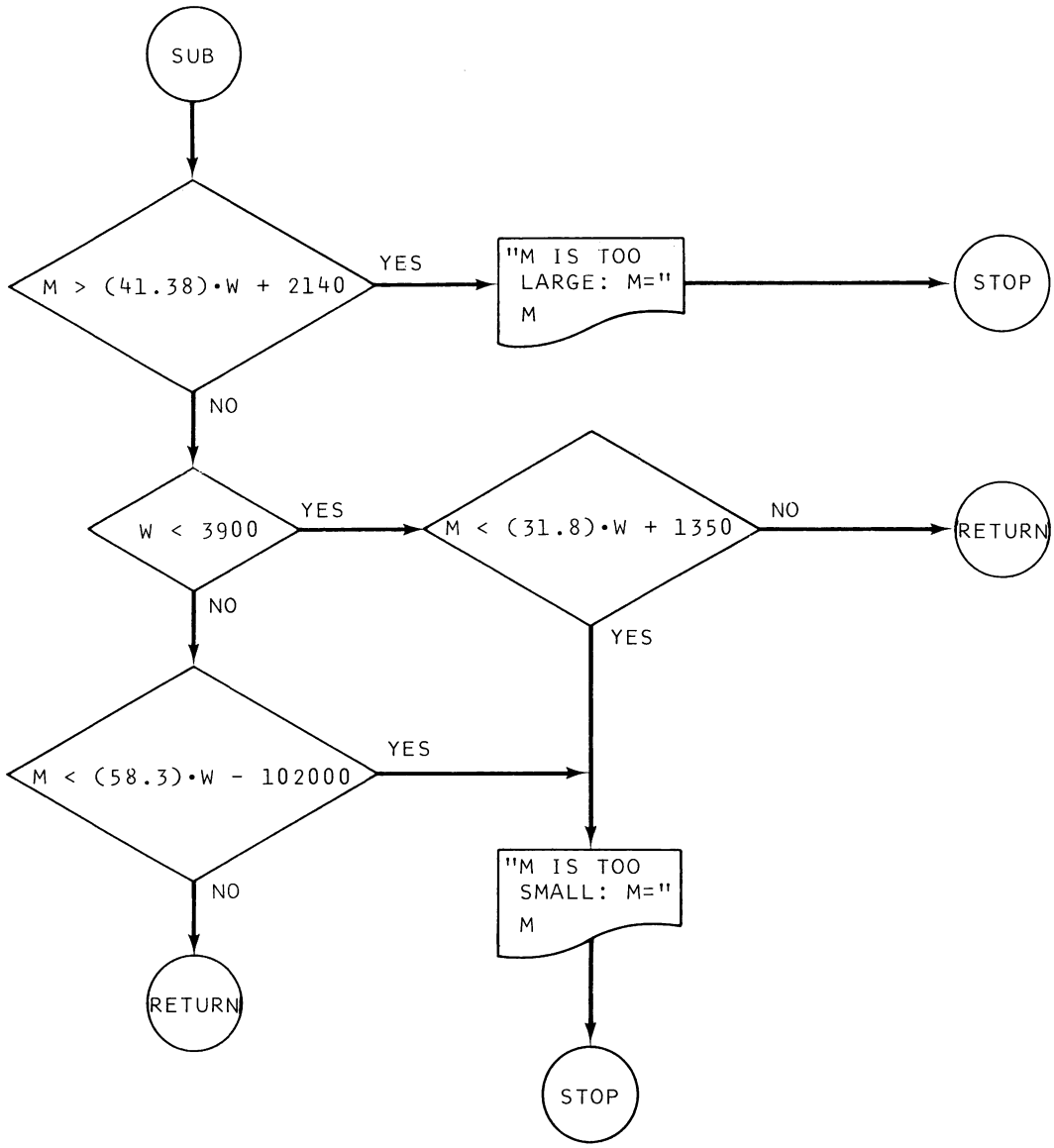


Figure II-5. Subroutine for the aircraft loading problem

You can check your program against mine if you wish by looking in Appendix IV. Versions in both BASIC and FORTRAN IV are presented there.

In sum, a flowchart or some other revelation of the logic is a necessity if a complex program is to be understood by others. You may not even want this (!), but a flowchart is still tremendously useful in requiring you to clarify and write down the logic for a program before writing it. It makes it possible for you to understand a program a year after you write it. The only reason that flowcharting isn't done or isn't done carefully enough is that one's fingers slip into gear so much easier than one's brain! *It is usually much easier to find a logic error in a flowchart than in a program!*

Whatever form it takes, I guarantee that a diagram of the logic in a complex program will also suggest ways to simplify and improve the program.

B. Practice Programs from Flowcharts

On the following pages (Figs. II-6 through II-10) are flowcharts for some interesting problems. Write programs from the flowcharts, and check them by verifying the sample answers given. You can start with Fig. II-3 for $\log_e (1 + X)$. Sample answers for this calculation are

```
X = .01,  Y = 9.95033E-3   AND   N = 4 TERMS
X = .1,   Y = 9.53102E-2   AND   N = 7 TERMS
X = .6,   Y = .470004      AND   N = 26 TERMS
```

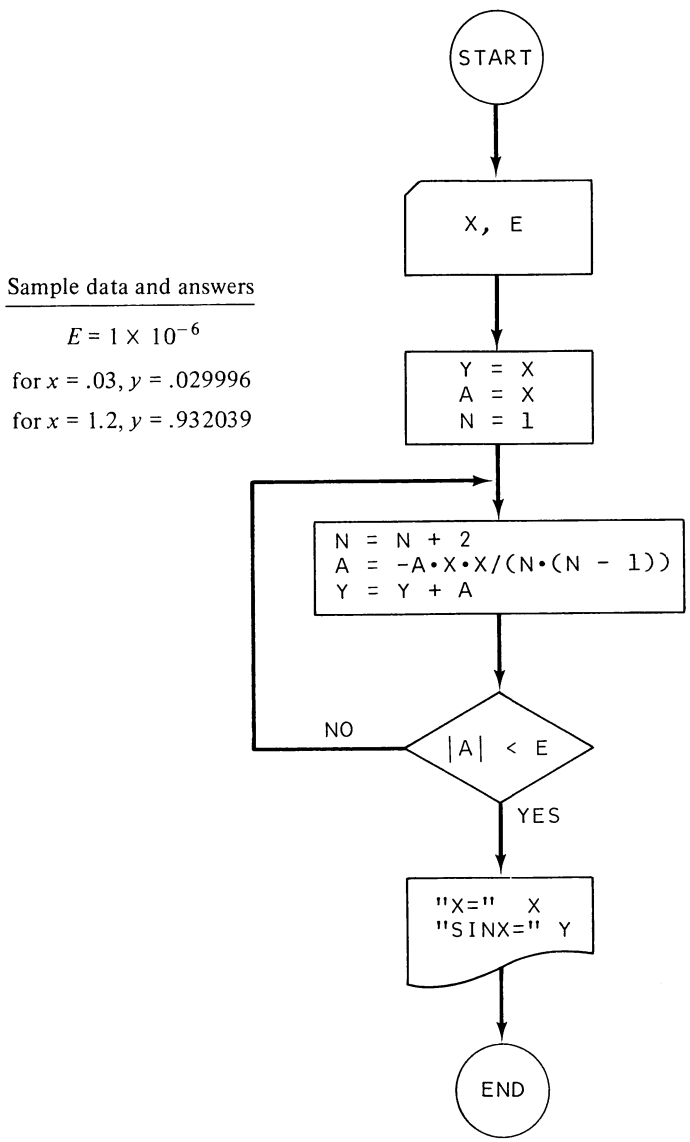


Figure II-6. Power series for $\sin(X)$

Sample data and answers

for $N = 6, X(1) = 4, X(2) = 7, X(3) = -4$
$X(4) = -7, X(5) = -5, X(6) = 9$
$S = -7, L = 9$

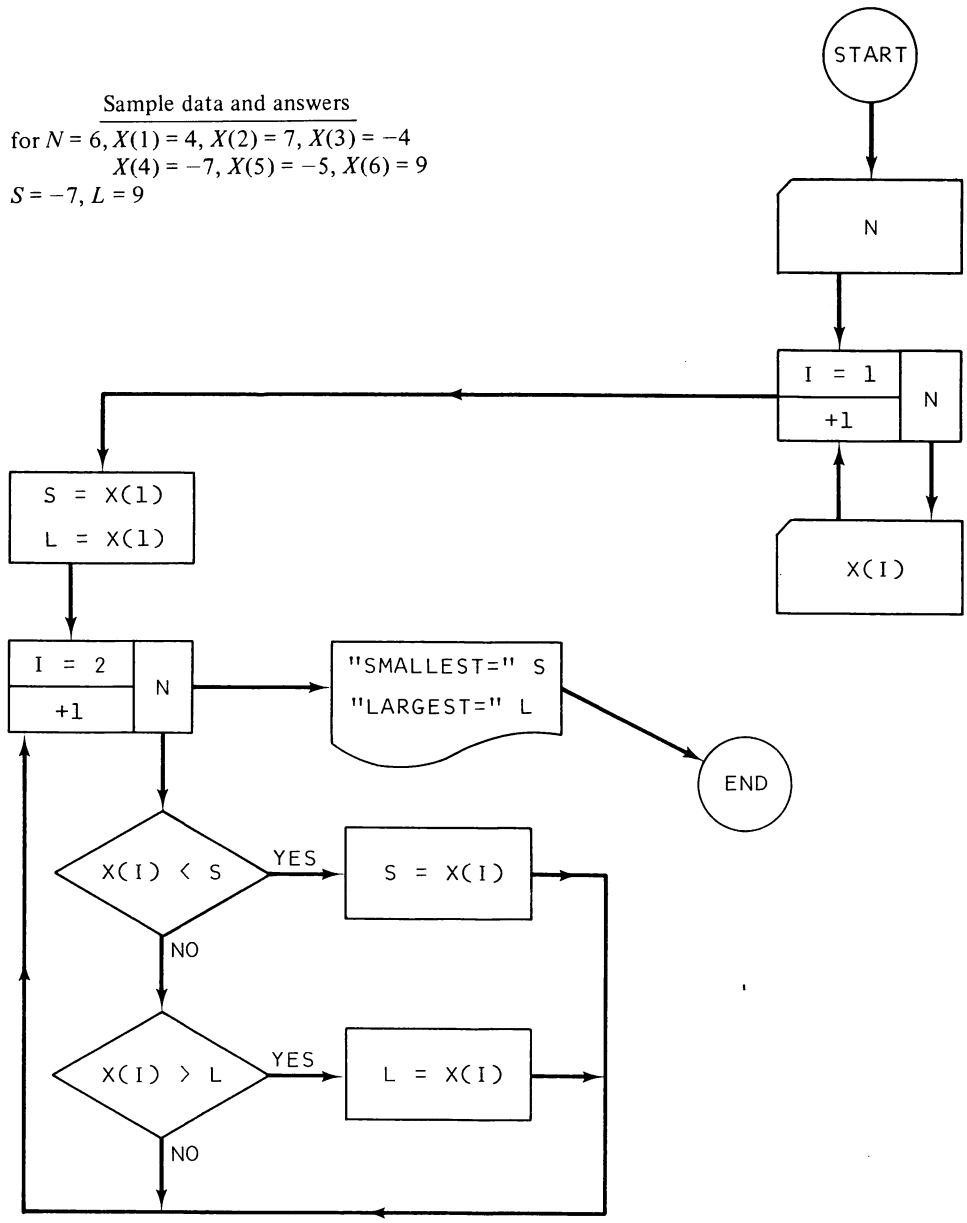
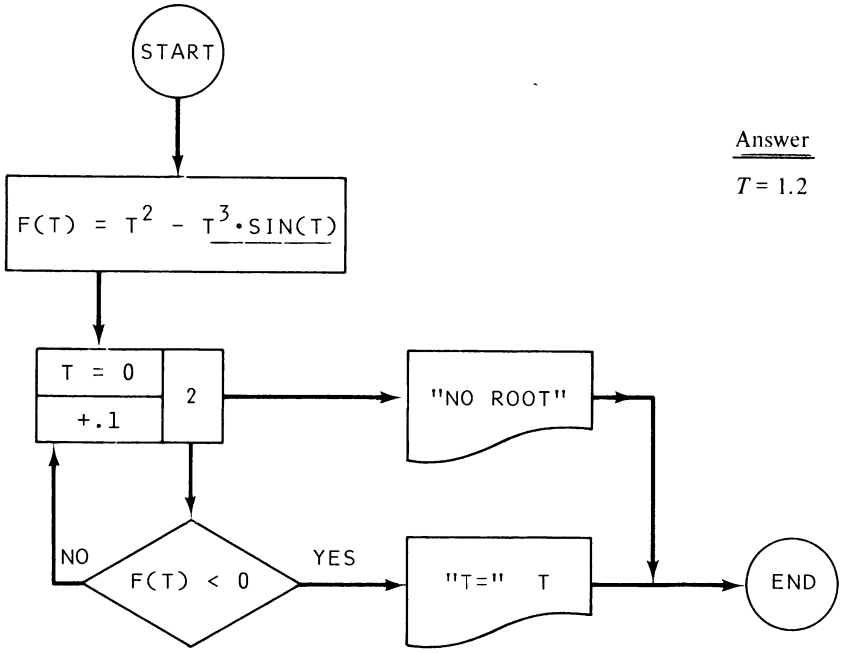


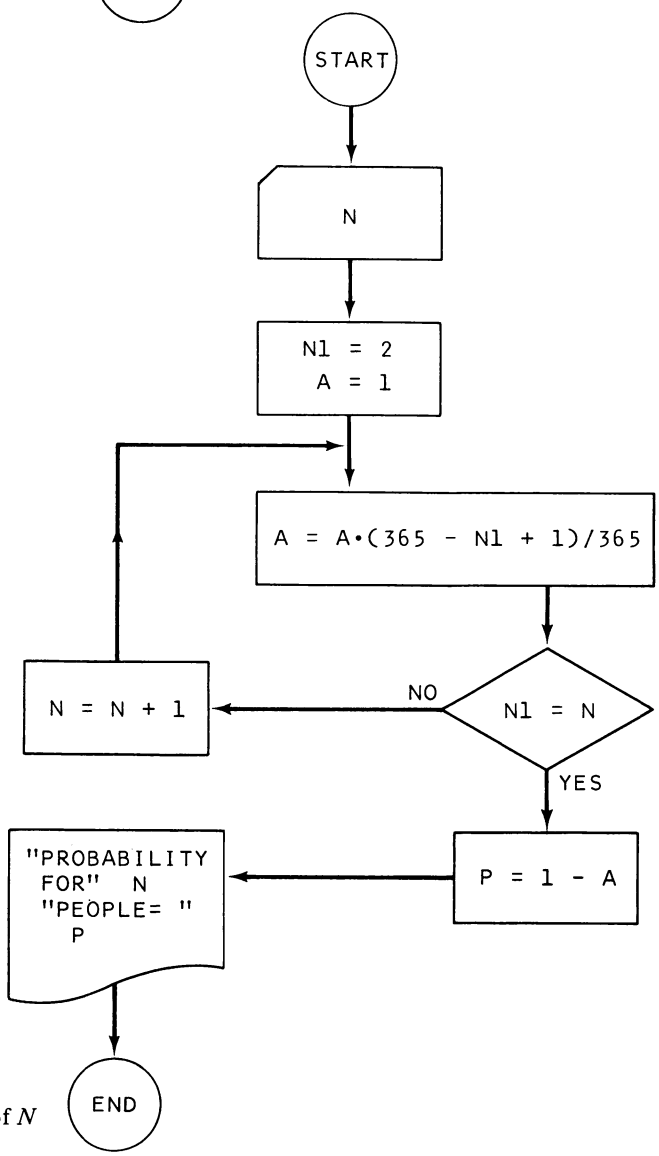
Figure II-7. Smallest and largest number in a set of numbers



Answer

$T = 1.2$

Figure II-8. Root of a function by successive substitution



Sample data and answers

$N = 2$, $P = 2.73973E\text{-}3$ ($\simeq .27\%$)
$N = 7$, $P = 5.62357E\text{-}2$ ($\simeq 5.6\%$)
$N = 20$, $P = .411438$ ($\simeq 41\%$)

Figure II-9. Probability that at least two people in a group of N people will have the same birthdate

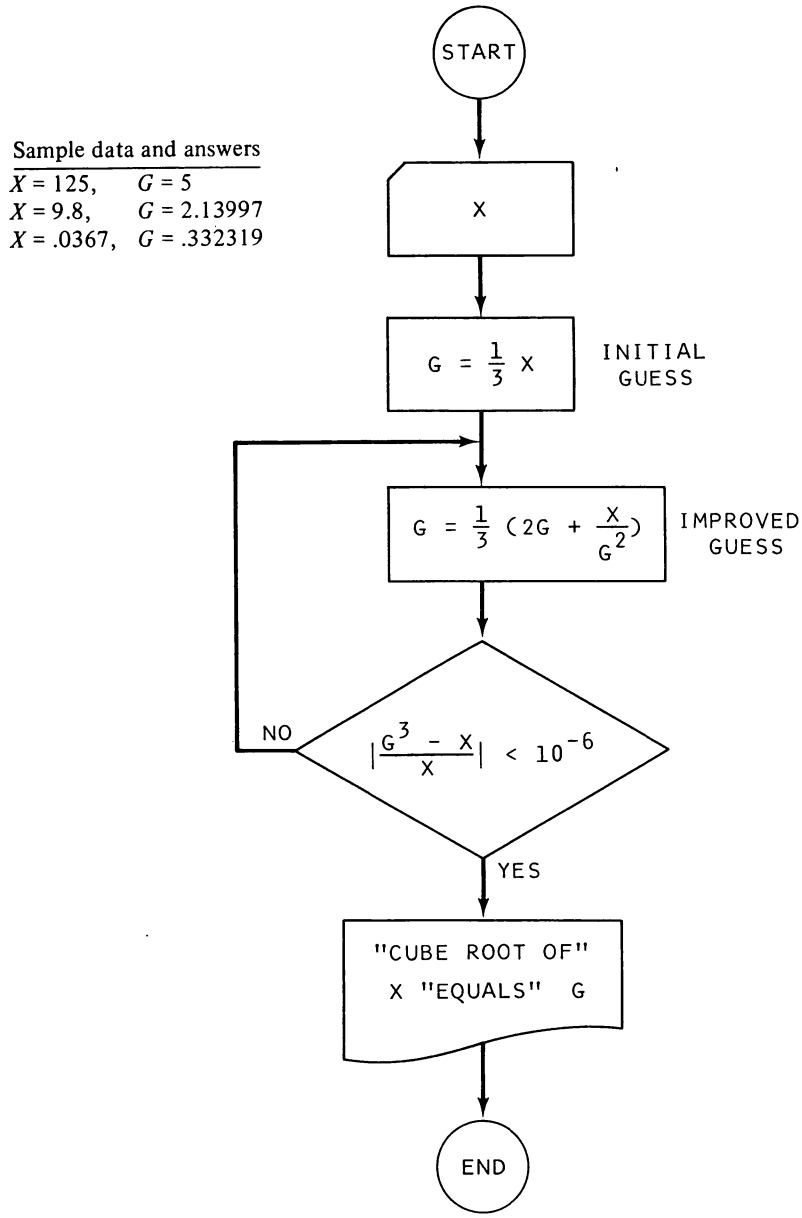


Figure II-10. Cube root by Newton's iteration method

C. Practice Flowcharts

Your drawing pencil should be sharp by now, so try drawing some flowcharts yourself. Use arrows to make the logical flow unambiguous. (Yes, I've given you my versions in an appendix, but a pox on you if you look ahead before trying to draw them yourself.)

1. Prob. 2, Chap. I, Sec. F. (This uses an IF statement for looping.)
2. Prob. 3, Chap. I, Sec. G.
3. Prob. 4, Chap. I, Sec. G.
4. Prob. 2, Chap. I, Sec. H.

5. Prob. 3, Chap. I, Sec. H.
6. Prob. 4, Chap. I, Sec. H.
7. Prob. 1, Chap. I, Sec. I.
8. Prob. 4, Chap. I, Sec. I.
9. Prob. 3, Chap. I, Sec. J.

III

Programming for Science and Engineering

A. Roots of Equations

Surprisingly often the analysis of a physical problem leads to an equation in a single unknown, and the answer we seek is the value(s) of the unknown that satisfies that equation. That is, given a **Y** that is a function of **X**, we wish to find one or more values of **X** (the roots) that will make **Y** equal to zero. Introductory science and engineering courses usually treat only the problems that lead to simple functions so that the root or roots can be obtained exactly from algebra. More realistic and more useful problems, though, usually lead to more complex functions so that the roots must be determined via a numerical technique. Thus possession of a good root-solving program can open many new doors in the understanding of physical phenomena; unfortunately, exploitation of this possibility (particularly in teaching) has lagged behind the development of computers for mass information storage and retrieval. Computer science has grown up apart since its birth within the physical sciences. Hopefully this chapter will give you insight into the use of the computer to solve real problems while you significantly increase your proficiency as a programmer. Physical problems that require the use of these techniques are included in the problems of the next chapter.

A geometrical interpretation of the root problem is helpful here. Consider the simple function

$$Y = F(X) = X + 3\cos(X) - 1$$

which is plotted in Fig. III-1 for the range from about $X = -2$ to +5. Note that this equation actually has *three* roots; i.e., there are three values of X for which Y is zero. There is no analytical method for finding the exact location of these roots, but the "high-speed moron" nature of the computer can be harnessed to locate them within any reasonable degree of accuracy.

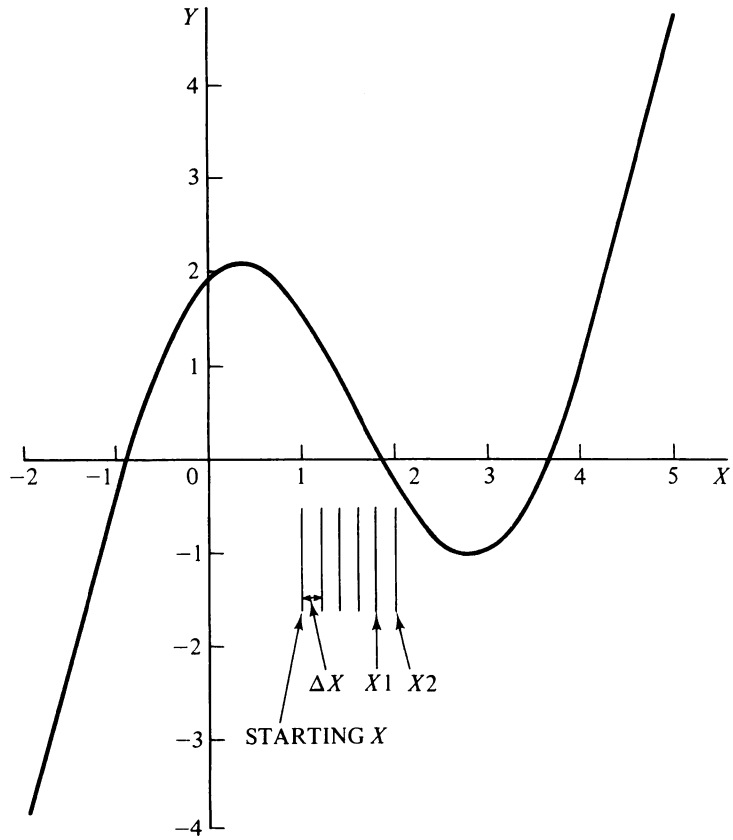


Figure III-1. Graph of the function $Y = F(X) = X + 3 \cos X - 1$

The real brute force method would be to realize that there is a root at about $X = -.9$ and start computing Y in, say, 0.01 increments near there. Having narrowed the root to a 0.01 interval then, one could repeat the process for 0.001 increments, and so on.

But a much faster method is to make an educated guess at the location of axis-crossing as soon as an interval in which Y changes sign is found. The *false-position method* does this by "drawing" a straight line between the two points and using that zero crossing for its next guess and does this over and over again. There are other root-solving methods that converge faster *sometimes*, but this method has proven to be useful in many different applications, and it is guaranteed to converge to the root when it has found a region where a continuous function changes sign.

In Figs. III-2 and III-3, I have drawn a flowchart for the false-position root-solving method. Follow along as I describe how it works; your understanding of the logic of the algorithm will enable you to debug your own program.

The part of the flowchart in Fig. III-2 is where a search is made to find a region where **F(X)** changes sign. The programmer must supply a starting value for X (X1), the size of the increment when searching for axis-crossing (ΔX), and an acceptable absolute error (E) in the located root. The equation is evaluated for this starting value, which gives Y1, and a check is made to see if we've stumbled on a root. If not, the value of X is incremented (X2 = X1 + ΔX), and again the equation is evaluated to obtain Y2. Another check is made for an accidental finding of a root; if not, a check is made to see if we've crossed the axis (in which case Y2

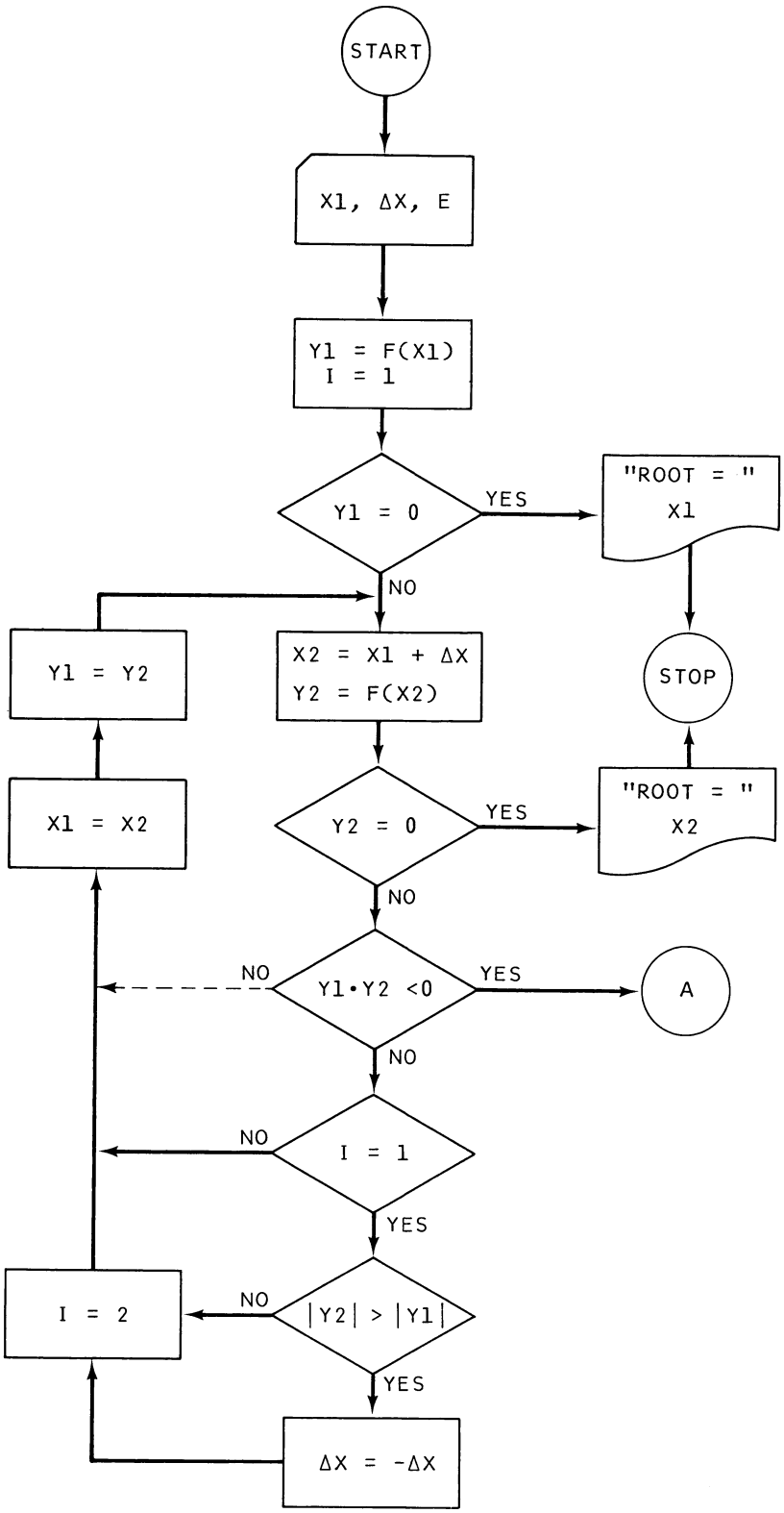


Figure III-2. Determination of an X such that $|Y| = |F(X)|$ is less than E (Part 1)

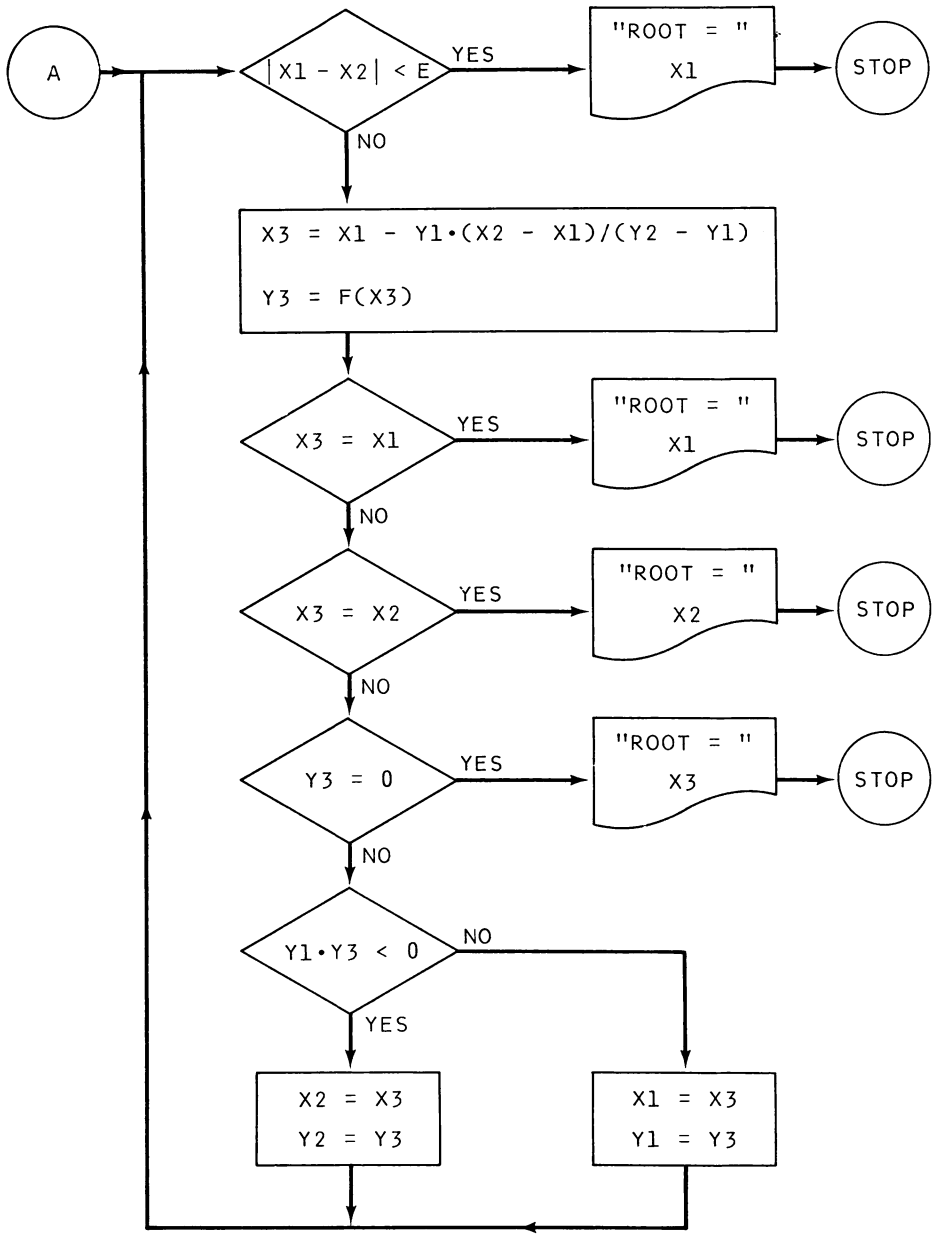


Figure III-3. Determination of an X such that $|Y| = |F(X)|$ is less than E (Part 2)

would have a different sign than Y1, and the product of Y1 and Y2 would be negative). The statement below this automatically reverses the direction of search ($\Delta X = -\Delta X$) on the first time through (I = 1) if Y2 is farther from the axis than was Y1; this can be bypassed by inserting a statement to direct execution along the dotted line. (This could be desirable if the function moves away from the axis and *then* moves back, as in Fig. III-1.)

If we haven't crossed the axis, then X is incremented until this happens. (Here we could get ourselves into an infinite loop if the function asymptotically approaches the axis but never gets there.) When axis-crossing does occur, then we know there must be a root between X1 and X2 *if* the function is continuous there. The starting value for X and these last two values of X1 and X2 are indicated on Fig. III-1 for a ΔX of 0.02.

The remaining task is to find the location of the axis crossing to within plus or minus E, and Fig. III-3 shows how this is done. Convergence here is quite rapid because the false position method estimates the location of the root (X3) based on an assumed straight-line variation between points X1 and X2 and then uses this estimate to choose a new pair of X1 and X2 such that their associated function values, Y1 and Y2, are again on opposite sides of the axis.

The formula for a straight line, given the locations of two points $(X1, Y1)$ and $(X2, Y2)$ on that line is

$$Y - Y_1 = \left(\frac{Y_2 - Y_1}{X_2 - X_1}\right) (X - X_1)$$

and the value of X for which this line passes through the axis is X_3, where

$$X_3 = X_1 - Y_1 \left(\frac{X_2 - X_1}{Y_2 - Y_1}\right)$$

by substitution with Y equal to zero, and thus the formula used in the flowchart has been derived.

The next two conditional checks (for $X1$ or $X2$ equal to $X3$) are necessitated by the nature of the computing process rather than by the numerical analysis theory. (These are perhaps the most baffling of the various types of bugs to find and eradicate.) If $X1$ is almost a root so that $Y1$ is very small but $Y2$ is much farther from the axis and therefore much larger, the second term in the equation for $X3$ can be so small (but not zero) that the computer stores the same number for $X3$ as for $X1$. For example, this would happen if there were seven-place accuracy and if $X1 = 1$, $Y1 = 1 \times 10^{-7}$, $X2 = 1.5$, and $Y2 = 25.32463$.

Similarly, the check for $X3$ being equal to $X2$ is required for the case where $X2$ is very close to the exact root and $X1$ is relatively far from it. Without this exit there would be no natural escape from the loop.

Finally, $X3$ is chosen as the new $X1$ if $Y3$ is on the same side of the axis as $Y1$; $X3$ is chosen as the new $X2$ if $Y3$ is on the other side of the axis from $Y1$.

Plenty of responsibility remains for the programmer even after the program is working properly, and this is where knowledge of the physical situation being modeled is invaluable. The programmer should choose a starting X and an increment ΔX such that axis-crossing occurs after a few iterations at most. E must be chosen based on theoretical and practical considerations, as a guide the choice of E equal to $(1 \times 10^{-6}/X)$ is a reasonable minimum for E if the computer accuracy is seven places and if X is the approximate value of the root.

A root-finding routine can be gainfully employed in some perhaps unexpected situations. Suppose we have the simultaneous equations

$$3X^2 - Y = 0$$
$$X + \log_e Y = 0$$

These can be solved for Y to give

$$Y = 3X^2$$
$$Y = -e^{-X}$$

and therefore an X that satisfies these equations is also a root of the equation

$$3X^2 + e^{-X} = 0$$

and for this the false-position program yields X = .458962 (and therefore Y = .631939) as a solution.

So write your own program now from the flowchart. You can check it with the following problems; the E used in each case was 1×10^{-6}.

Problems

1. Find the roots of the function of Fig. III-1. *Answer*: −.88947, 1.86236, 3.63796
2. Find a root of the following functions:
   (a) $Y = 1/X^3 + e^{-X}$ *Answer*: −.772883
   (b) $1.6 \log_e (|1.67X|) - 1.67X$ *Answer*: −.396054
   (c) $(X^2 - 4.8)^{1/3} - 4.8 \sin X$ *Answer*: 2.82937
   (d) $4.32X^2 - \sin(4.32X)$ *Answer*: .460076
3. Solve the following simultaneous equations.
   A. $\sin X + \tan Y = 3$
      $e^X + Y = 2$
      Answer: X = −.326095, Y = 1.27826
   B. $Y(1 - X)^{1/2} = 3X^2 - 2$
      $Y - \sin X = -e^X Y$
      Answer: X = .877404, Y = .225892
   C. $e\sqrt{X} - Y = 11$
      $\cos X - Y = -1$
      Answer: X = 5.67012, Y = −.182115

B. Simultaneous Equations

Simultaneous equations are commonly encountered in the mathematical description of physical phenomena; they result whenever there is an interdependence of two or more variables in determining an outcome. Perhaps the simplest example is the determination of the range of a projectile; the horizontal distance depends on the horizontal velocity and the time of flight, but the time of flight depends on the motion in the vertical plane.

The goal for this section is to provide a routine that quickly and efficiently solves up to ten or so simultaneous linear equations. There are some direct applications to problems (particularly electronic circuit theory in which each new branch adds another equation), but our primary use of the routine will be in the curve-fitting program of Sec. C.

Gauss-Jordan elimination is a standard technique for the solution of a group of linear simultaneous equations. (The equations are assumed to be linear—i.e., of the first degree—in the unknowns, and it is assumed that there are as many equations as there are unknowns.) Gauss-Jordan accomplishes its task by manipulating each of the equations in such a way that one of the unknowns is eliminated from all but one of the equations and by doing this repeatedly.

For example, suppose we start with a set of *two* simultaneous equations in the general form

$$A(1,1) \cdot X1 + A(1,2) \cdot X2 = A(1,3)$$
$$A(2,1) \cdot X1 + A(2,2) \cdot X2 = A(2,3)$$

Specifically, consider

$$2 \cdot X1 + 4 \cdot X2 = -4$$
$$3 \cdot X1 + 10 \cdot X2 = 2$$

We can easily solve these by substitution to find that $X1 = -6$ and $X2 = 2$. But with more than two equations, the substitution method becomes much too tedious. However, there are rules that say that we can (a) change the order of the equations, (b) multiply any equation by a constant, and (c) multiply one equation by a constant and add to any other equation, all without changing the solution. The Gauss-Jordan method uses these rules to make the diagonal elements [$A(1,1)$, $A(2,2)$, etc.] equal to *one* and the other coefficients equal to *zero*, so that the far right column is then the listing in order of the solutions for the unknowns. The two steps for the previous equations are the following:

$$2 \cdot X1 + 4 \cdot X2 = -4$$
$$3 \cdot X1 + 10 \cdot X2 = 2$$
$$\downarrow$$
$$X1 + 2 \cdot X2 = -2$$
$$0 + 4 \cdot X2 = 8$$
$$\downarrow$$
$$X1 + 0 = -6$$
$$0 + X2 = 2$$

In the first step, the first equation was divided by the first-row diagonal element (2); in the second step, the second equation was divided by the second-row diagonal element (4). Thus the method will certainly fail if any of the diagonal elements are or become zero. This doesn't necessarily mean that no solution exists; in fact, theory suggests that the greatest accuracy is attained when the diagonal element being used has the largest value and when equations below it can be exchanged with it to accomplish this. However, I have found no consistent improvement in accuracy in doing this, no doubt due to an accumulation of round-off error from the additional operations, and these additional operations do add significantly to the execution time. Also, zero diagonal elements do not appear in real problems involving curve-fitting. Thus the algorithm to be given will be the simple and efficient one illustrated previously.

Figures III-4 and III-5 are flowcharts of the method. Figure III-4 shows how the coefficients of the unknowns are supplied to the program. The first statement tells the computer how much memory is required; the coefficients are stored in the subscripted variable **A** which must be dimensioned **A(N2,N2+1)**, where **N2** is the number of equations. **A(10,11)** is indicated because ten tends to be a practical maximum for seven- or eight-digit accuracy. For the foregoing example, the input data are

$N2 = 2$
$A(1,1) = 2$
$A(1,2) = 4$
$A(1,3) = -4$
$A(2,1) = 3$
$A(2,2) = 10$
$A(2,3) = 2$

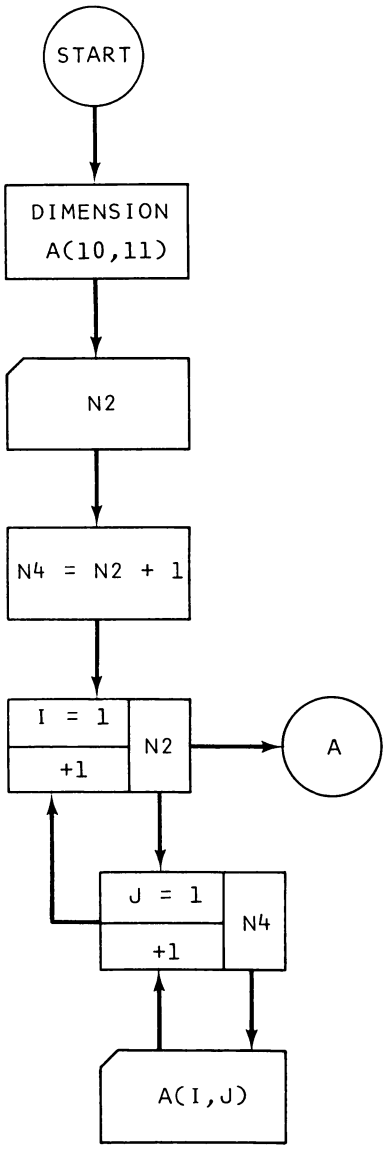


Figure III-4. Solution of a group of $N2$ simultaneous equations (Part 1)

Figure III-5 shows how the elimination steps illustrated earlier are implemented in general. The diagonal element $[P = A(K,K)]$ is divided into each coefficient for each row (K) in turn, as indicated earlier.

This is a good time to reassure you that with patience you can debug even the most intimidating program. Try following me through the flowchart to see that our original equations really are transformed in the expected fashion.

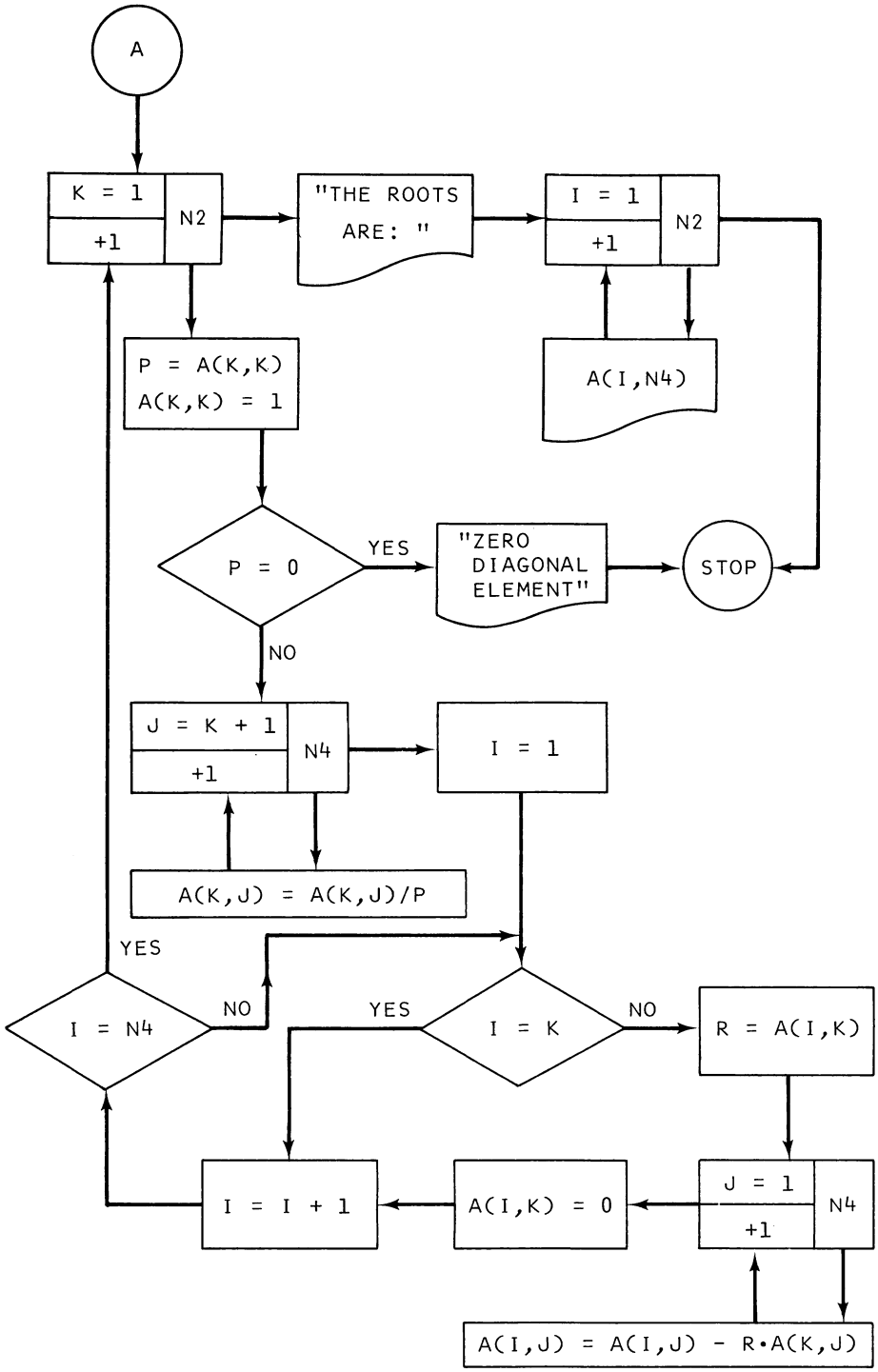


Figure III-5. Solution of a group of $N2$ simultaneous equations (Part 2)

$\underline{K = 1}$

$P = A(1,1) = 2$

$A(1,1) = 1$

$\underline{J = 2}$

$$A(1,2) = \frac{A(1,2)}{P} = \frac{4}{2} = 2$$

$\underline{J = 3}$

$$A(1,3) = \frac{A(1,3)}{P} = -\frac{4}{2} = -2$$

$\underline{I = 1}$

$\underline{I = 2}$

$R = A(2,1) = 3$

$\underline{J = 1}$

$$A(2,1) = A(2,1) - R \cdot A(1,1)$$
$$= 3 - 3 \quad (1) = 0$$

$\underline{J = 2}$

$$A(2,2) = A(2,2) - R \cdot A(1,2)$$
$$= 10 - 3 \quad (2) = 4$$

$\underline{J = 3}$

$$A(2,3) = A(2,3) - R \cdot A(1,3)$$
$$= 2 - 3 \ (-2) = 8$$

So the first step has indeed been accomplished. You should trace through the flowchart for $K = 2$ to check that the second step is likewise accomplished. This should give you enough information to debug your program; you can print out intermediate results at any point and compare them with these hand-calculated results until the bug is uncovered. It is suggested that you write your program in such a way that it can easily be made into a subroutine for use in Sec. C.

Problems: Solve the following simultaneous equations.

1. $3 \cdot X1 + 4 \cdot X2 = 8$
   $4 \cdot X1 + 5 \cdot X2 = 12$
   Answer: $X1 = 8$, $X2 = -4$

2. $X1 + 2 \cdot X2 + 3 \cdot X3 = 4$
   $2 \cdot X1 - X2 + 4 \cdot X3 = 7$
   $-X1 + 3 \cdot X2 - X3 = 2$
   Answer: Zero diagonal element

3. $7 \cdot X1 + 2 \cdot X2 + 3 \cdot X4 = 20$
   $2 \cdot X1 - 3 \cdot X2 - 2 \cdot X4 = 10$
   $3 \cdot X1 + X2 - 4 \cdot X4 = -2$
   Answer: $X1 = 2.81481$, $X2 = -2.74074$
   $X3 = 1.92593$

4. $X1 + X2 + X3 + X4 = 10$
   $X1 - X2 + X3 - X4 = 12$
   $2 \cdot X2 + X3 = 8$
   $X1 - 2 \cdot X4 = 5$

   Answer: $X1 = 2$, $X2 = -.5$, $X3 = 9$
   $X4 = -.5$

5. $4 \cdot X1 - X2 + 3 \cdot X3 - X4 = 0$
   $2 \cdot X1 + 3 \cdot X2 - 4 \cdot X3 + X4 = 12$
   $X1 - X2 + X3 - 2 \cdot X4 = -3$
   $3 \cdot X1 + 2 \cdot X2 - 2 \cdot X3 + 3 \cdot X4 = 5$

   Answer: $X1 = -.5625$, $X2 = 9.3125$
   $X3 = 3.25$, $X4 = -1.8125$

C. Least Squares Fit of Data to a Polynomial

One of the most useful of all computer programs for the scientist or engineer is a general program that will fit data to a polynomial of arbitrary degree $N1$, a polynomial of the form

$$Z(I) = A0 + A1 \cdot X(I) + A2 \cdot X(I)^2 + A3 \cdot X(I)^3 + \ldots + AN1 \cdot X(I)^{N1}$$

Suppose the data consist of N data pairs, $X(I)$ and $Y(I)$, where I is equal to 1 through N. Then the problem is to choose the coefficients $A0, A1, A2, \ldots, AN1$ in such a way as to obtain the "best" correspondence of the $Z(I)$ to the $Y(I)$. The easiest technique mathematically is to minimize the sum of the *squares* of the distance between the polynomial $Z(I)$ and the data points (because we want only the absolute value of the distance to be a minimum, without regard to the sign of the distance).

Thus, if we have four data points, as shown in Fig. III-6, we wish to minimize the sum

$$S = \Sigma\, d_i^2 = \Sigma\, \{Z(I) - Y(I)\}^2$$
$$= \Sigma\{A0 + [A1 \cdot X(I)] + [A2 \cdot X(I)^2] + [A3 \cdot X(I)^3] + \ldots + [AN1 \cdot X(I)^{N1}] - Y(I)\}^2$$

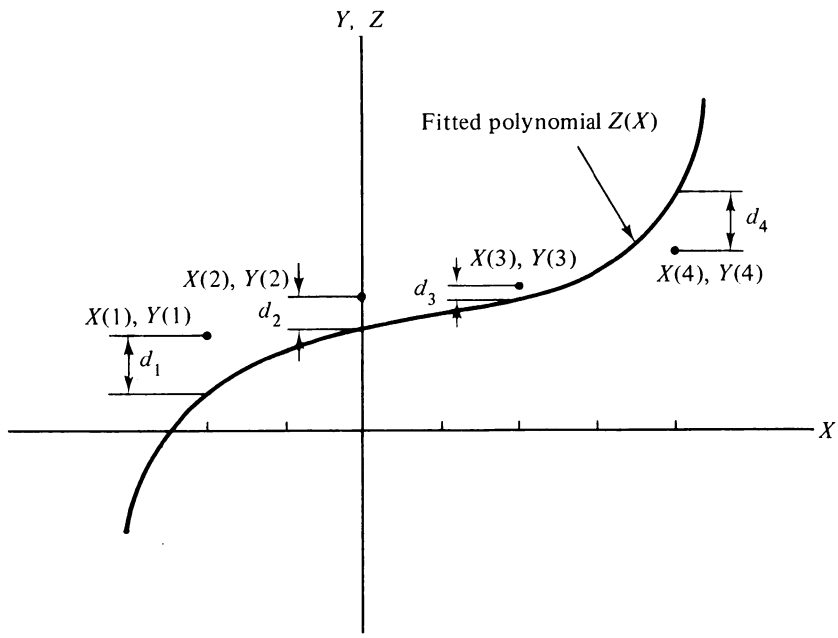


Figure III-6. Four data points and a fitted curve

where the summation index i goes from 1 to 4.

From the calculus we can hope to obtain the best value for the coefficients $A0$ through $AN1$ by calculating the partial derivatives of S with respect to each of the coefficients and by separately equating each to zero. For example, for $A0$

$$\begin{aligned}
&\frac{\partial}{\partial A0} \Sigma\{Z(I) - Y(I)\}^2 \\
&\quad = \frac{\partial}{\partial A0} \Sigma\{A0 + [A1 \cdot X(I)] + [A2 \cdot X(I)^2] + [A3 \cdot X(I)^3] + \ldots + [AN1 \cdot X(I)^{N1}] \\
&\qquad - Y(I)\}^2 \\
&\quad = 2 \Sigma\{A0 + [A1 \cdot X(I)] + [A2 \cdot X(I)^2] + [A3 \cdot X(I)^3 + \ldots + [AN1 \cdot X(I)^{N1}] \\
&\qquad - Y(I)\} \\
&\quad = 0
\end{aligned}$$

where now the summation is from 1 to N.

We can rearrange this equation to get

$$\Sigma\{A0 + [A1 \cdot X(I)] + [A2 \cdot X(I)^2] + [A3 \cdot X(I)^3] + \ldots + [AN1 \cdot X(I)^{N1}]\} = \Sigma Y(I)$$

or

$$[N \cdot A0] + A1 \cdot \Sigma X(I) + A2 \cdot \Sigma X(I)^2 + A3 \cdot \Sigma X(I)^3 + \ldots + AN1 \cdot \Sigma X(I)^{N1} = \Sigma Y(I)$$

Similarly, taking the partial derivative with respect to $A1$ gives us

$$A0 \cdot \Sigma X(I) + A1 \cdot \Sigma X(I)^2 + A2 \cdot \Sigma X(I)^3 + \ldots + AN1 \cdot \Sigma X(I)^{N1+1} = \Sigma Y(I) \cdot X(I)$$

and the partial derivative with respect to $AN1$ gives us

$$A0 \cdot \Sigma X(I)^{N1} + A1 \cdot \Sigma X(I)^{N1+1} + \ldots + AN1 \cdot \Sigma X(I)^{2 \cdot N1} = \Sigma[Y(I) \cdot X(I)^{N1}]$$

Thus we end up with a set of $N1+1$ simultaneous equations in the $N1+1$ coefficients $A0, A1, A2, \ldots, AN1$, which we can solve with the help of the program in Sec. B. If we carefully look at the multiplying constants, we discover that many of them are the same; in fact, there are just $2 \cdot N1+1$ terms to be calculated on the left-hand side of the equations and just $N1+1$ terms on the right-hand side. Each term involves some product of the $X(I)$ data values, so those of higher degree can be obtained from those of lower degree to minimize the number of multiplications.

The general plan for the calculation of the coefficients for these simultaneous equations is shown in Fig. III-7. The coefficients are first calculated under the variable Z, and the second block shows how many of these coefficients are the same and therefore need to be calculated only once. Then the coefficients are transferred into the doubly subscripted variable A, where they are ready to be sent to the simultaneous equation subroutine for solution.

A flowchart for the calculation of the coefficients is given in Figs. III-8, III-9, and III-10. (You could have been asked to devise your own logic for this process, but this flowchart supplies a well-developed algorithm in the interests of giving you an efficient and very usable program. Your need to understand how it works remains.)

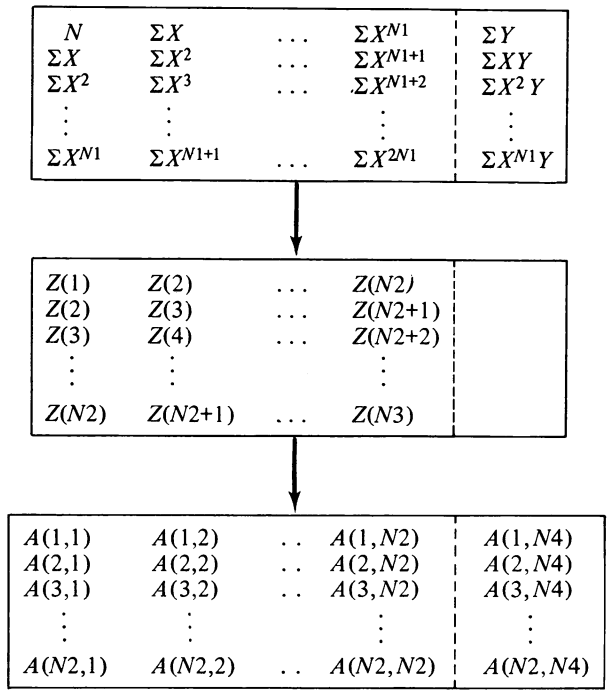


Figure III-7. Block diagram of the calculation of the elements of the array of simultaneous equations for a least squares fit to a polynomial of degree $N1$

Figure III-8 shows how the data (N pairs) are read into the program and how the programmer then selects the degree of the polynomial ($N1$) that he wishes to fit to his data. The number of terms in the polynomial ($N2$) is one more than the degree because of the constant term. Thus if $N1 = 2$, the fit is to $Y = A0 + (A1 \cdot X) + (A2 \cdot X^2)$. The last part of the chart sets some variables, to be used later in summing, equal to zero.

The coefficients are calculated from the data in Fig. III-9. The computation begins with the first element of the far right-hand column ($A1$, $N4$), which is simply a sum of the Y values. Then the variable Z is used to save the computation of various powers of X; the variable B is used as an intermediate variable so that a higher power can be calculated from the one below it. The remaining elements of the far right-hand column, $A(J,N4)$, are calculated at the bottom of this loop.

Finally, in the last nested loop, all of the final array A, except for the far right-hand column, is defined from the variable Z.

Execution then transfers to the Gauss-Jordan routine described in Sec. B. Recall that this routine returns with the unknowns stacked in the far right-hand column of A, $A(I,N4)$. Thus for $N1$ equal to 2, $A0$ is in $A(1,3)$, $A1$ is in $A(2,3)$, and $A2$ is in $A(3,3)$.

Figure III-10 shows the answers being printed out and an evaluation being made of the degree of success of the fitting operation. One evaluation is obtained by using the fitted curve (polynomial) to calculate a Y ($Y2$ in the program) from each $X(I)$ and by printing out this calculated value along with each supplied $Y(I)$. $C2$ is used to store the sum of the deviations squared in the same loop and is therefore just the sum that we have attempted to minimize. Before printing out this second evaluation, though, it is divided by the factor $(N - N2)$. This factor makes the sum more meaningful by increasing it if the number of

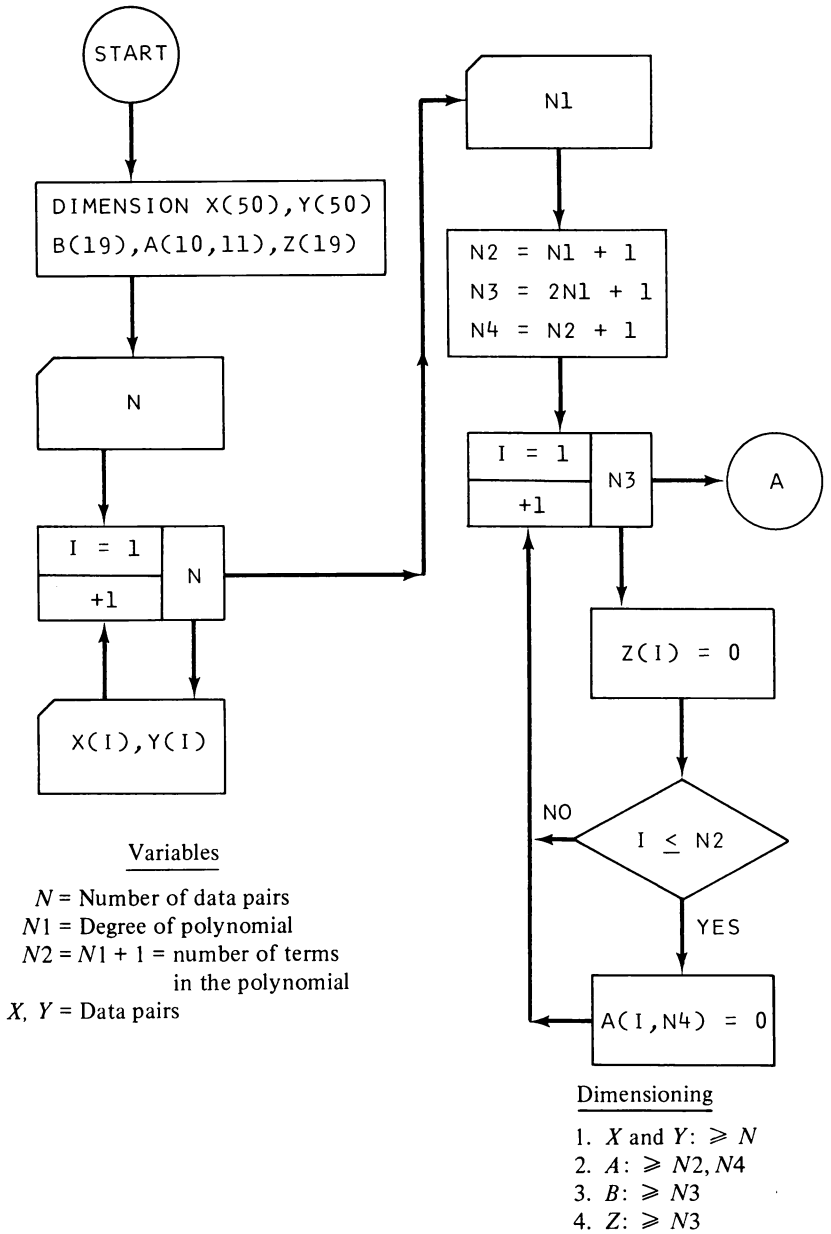


Figure III-8. Least squares fit to polynomial of degree $N1$ (Part 1)

adjustable constants ($N2$) in the polynomial is anywhere close to the number of data pairs (N). Specifically, it makes the sum a better indicator of whether the data actually could have been produced by the same functional dependence; for example, if you supply only two data pairs, you can do a perfect job of fitting a straight line through these two points, but you shouldn't claim that your data prove a direct proportionality between dependent and independent variables! In fact, you can make no claim at all, and the correction factor reflects this fact by producing an overflow number for the adjusted sum.

The adjusted sum is often referred to as χ^2 (chi squared)—or better known as the *reduced chi squared* because of the adjustment. When we take the square root of this sum,

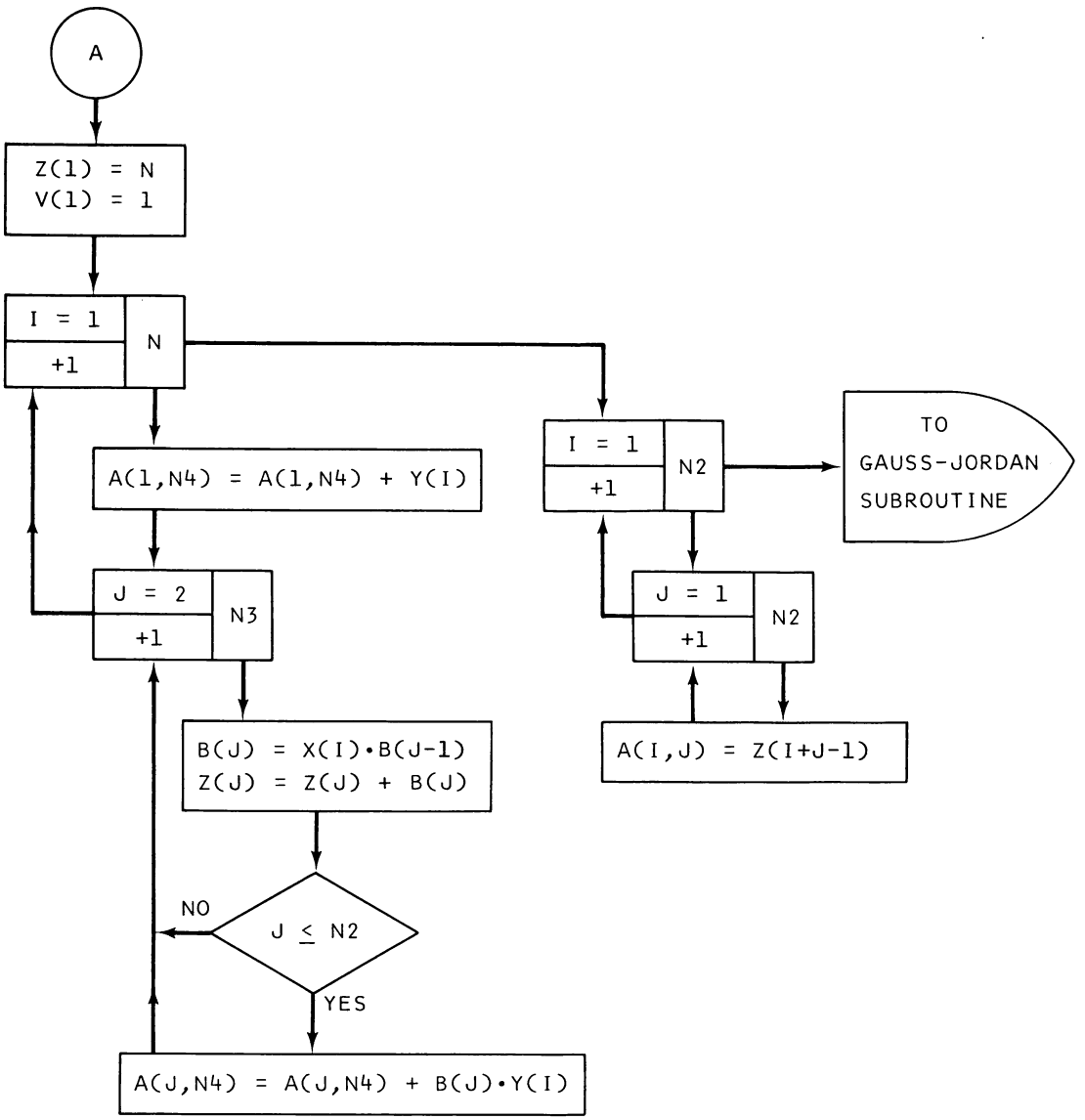


Figure III-9. Least squares fit to polynomial of degree $N1$ (Part 2)

we have a good indication of the average vertical distance between the curve and the data. If you have a plotter available, you surely will want to plot both the curve and the data for visual evidence of a good fit. Also, if you are using an interactive computer language at a terminal, you may want to add a statement to send execution back for a new try with a higher-degree polynomial without having to reenter the data.

Two data pairs and a first-degree polynomial do provide a good way to check your program (but plan to bypass the calculation of χ^2). For example, if you try

$$X(1) = 0, \qquad Y(1) = 1$$
$$X(2) = 1, \qquad Y(2) = 2$$

and $N1$ equal to 1, you should get the polynomial $Y = X + 1$. To see why this is so, follow

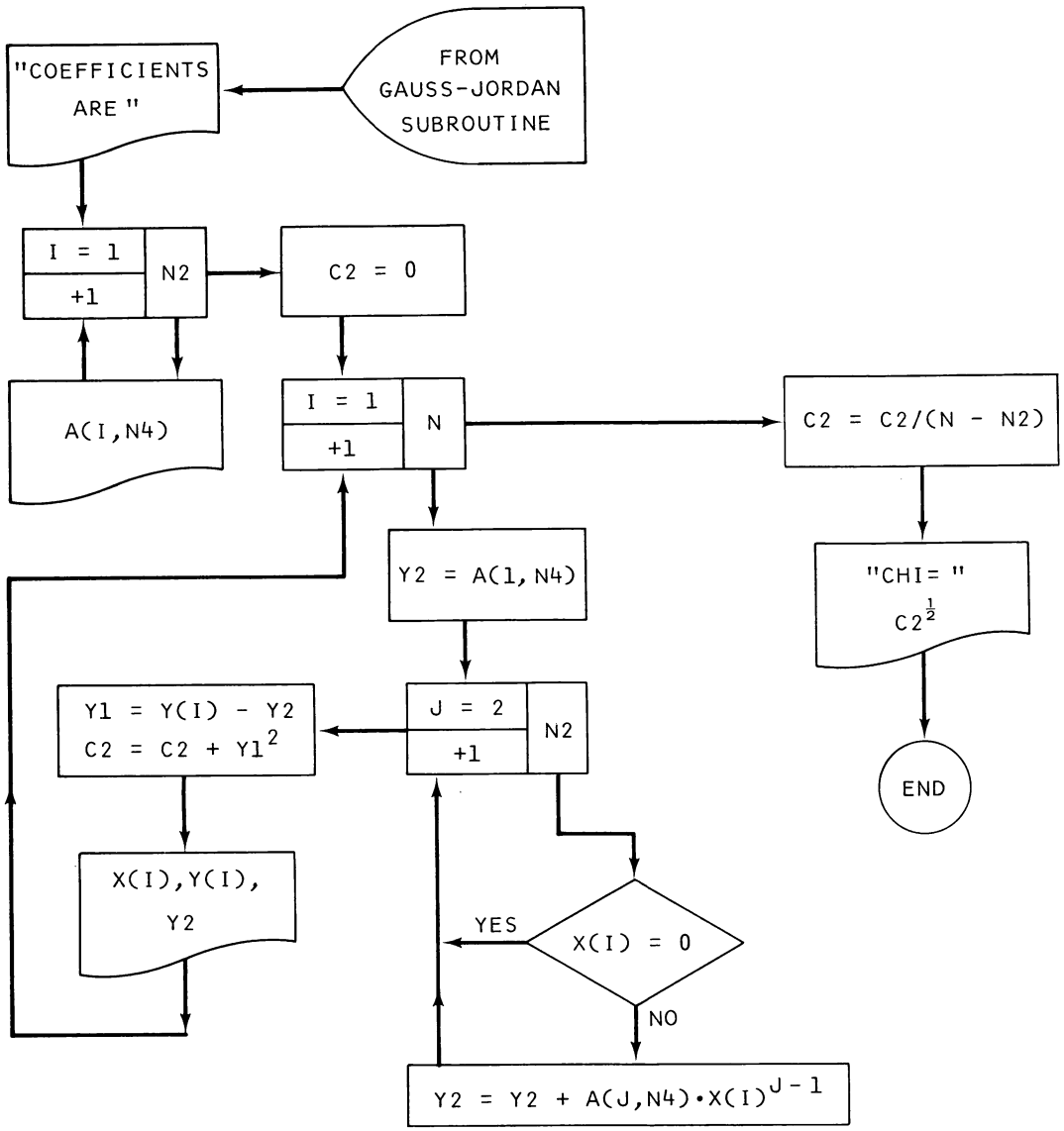


Figure III-10. Least squares fit to polynomial of degree $N1$ (Part 3)

me through the calculation:

The components of the array should be

$A(1,1) = N = 2$
$A(1,2) = \Sigma X = 0 + 1 = 1$
$A(1,3) = \Sigma Y = 1 + 2 = 3$
$A(2,1) = \Sigma X = 1$
$A(2,2) = \Sigma X^2 = 0 + 1 = 1$
$A(2,3) = \Sigma XY = (0)(1) + (1)(2) = 2$

and therefore the simultaneous equations to be solved are

$$2 \cdot A0 + A1 = 3$$
$$A0 + A1 = 2$$

which, by substitution, have the solution $A0 = 1, A1 = 1$, so that the polynomial must be $Y = X + 1$.

Another excellent check is to generate some data from a known polynomial. For example, from

$$Y = 4.7 - X + 1.8X^2$$

we get the set of data

$X(1) = -5, \quad Y(1) = 54.7$
$X(2) = -3, \quad Y(2) = 23.9$
$X(3) = -1, \quad Y(3) = 7.5$
$X(4) = 1, \quad Y(4) = 5.5$
$X(5) = 3, \quad Y(5) = 17.9$

and for $N1$ equal to 2, the program yields $A0 = 4.7, A1 = -1$, and $A2 = 1.8$; χ is computed as 6.48569E−06 rather than zero because of round-off error.

Finally, try many numbers:

$X(1) = 1.04, Y(1) = -2.26$	$X(11) = .74, Y(11) = -3.85$
$X(2) = 1.01, Y(2) = -2.49$	$X(12) = .71, Y(12) = -3.96$
$X(3) = .98, Y(3) = -2.69$	$X(13) = .68, Y(13) = -4.07$
$X(4) = .95, Y(4) = -2.88$	$X(14) = .65, Y(14) = -4.17$
$X(5) = .92, Y(5) = -3.05$	$X(15) = .62, Y(15) = -4.27$
$X(6) = .89, Y(6) = -3.21$	$X(16) = .59, Y(16) = -4.37$
$X(7) = .86, Y(7) = -3.36$	$X(17) = .56, Y(17) = -4.48$
$X(8) = .83, Y(8) = -3.49$	$X(18) = .53, Y(18) = -4.58$
$X(9) = .80, Y(9) = -3.62$	$X(19) = .50, Y(19) = -4.68$
$X(10) = .77, Y(10) = -3.74$	$X(20) = .47, Y(20) = -4.78$

From this data I obtain

$N1 = 2$: $A0 = -5.07275, A1 = -.839727, A2 = 3.34486, \chi = .0340498$
$N1 = 3$: $A0 = -8.17696, A1 = 12.3456, A2 = -14.6797, A3 = 7.9556,$
$\chi = 5.30277E-03$
$N1 = 4$: $A0 = -8.73608, A1 = 15.4958, A2 = -21.1696, A3 = 13.7583,$
$A4 = -1.90322, \chi = 6.64410E-3$
$N1 = 5$: $\chi = .0347066$ (5 second execution time)
$N1 = 6$: $\chi = 3.21365E-03$ (8 second execution time)
$N1 = 8$: $\chi = 6.70281E-03$ (10 second execution time)

Is it possible to guess the polynomial by which the data were generated? No, not in this case, because the polynomial was

$$Y = -7.04 + 6.70X - 5.67X^2 + 4.57X^3 - 3.45X^4 + 2.34X^5$$

Three significant figures in the data simply aren't adequate with a polynomial of such high degree to determine the originating polynomial, even with no scatter in the data. It's interesting to note that χ reaches a minimum for $N1 = 3$ and then a second minimum after $N1 = 6$. The first minimum, though, is about 0.005, and the supplied data was good to only ±0.005, so that should have warned us that an attempt to obtain a smaller χ would probably not be meaningful.

D. Nonlinear Least Squares Fit of Data to an Arbitrary Function

Theoreticians like to predict the functional form for physical measurements of all kinds. Experimentalists then like to fit their measurements to the predicted function and thereby provide determination of the fundamental parameters involved in the process. In the vast majority of cases, the parameters do not enter linearly into the functional form, and it is of particular interest to develop a general technique for handling such problems.

Simplest among such techniques (and often the least efficient) is one that keeps changing the parameters, one by one, until the average deviation of the data points from calculated values is arbitrarily small. Because the parameters usually are quite dependent on one another, the optimization for each usually must be made many times. As a result the process can require a lot of computer time, but the generality and simplicity of the approach still make it quite attractive.

Suppose the functional form is

$$F(X) = A(1) \cdot \sin [(4 \cdot X) + A(2)]$$

Then the procedure is to supply starting values for $A(1)$ and $A(2)$ and starting values for step changes to them, $D(1)$ and $D(2)$. The average deviation of the data points from the functional form is calculated, then $A(1)$ is altered until a value is found that minimizes the average deviation, and then $A(2)$ is altered to find the value for it that minimizes the average deviation again. The fit is continued until the fit satisfies the programmer's criterion—or until someone gets tired.

In order to find the value of $A(1)$ that minimizes the average deviation, the average deviation is calculated for $A(1) = A(1) + D(1)$; if that decreases the average deviation, then $A(1)$ is changed by the same amount again, and this is continued until the average deviation starts to increase. A parabola is then fitted to the last three points, and the minimum of the parabola gives the new $A(1)$. However, if $A(1) = A(1) + D(1)$ produced an increasing average deviation, then $D(1)$ is changed to $-D(1)$, and incrementation is continued in this direction until a local minimum in the average deviation can be located as before. The same procedure is also followed with $A(2)$.

Refer to Fig. III-11 to see how the minimum of a parabola can be obtained from knowledge of three points on the parabola. [We use a parabola to represent the variation of average deviation with $A(1)$ or $A(2)$, because this is the simplest polynomial that shows the expected minimum surrounded by at least local maxima.] A parabola with a minimum at $(X0,Y0)$ is described by the equation

$$Y - Y0 = P(X - X0)^2$$

where the constant P is a measure of how rapidly Y increases away from the minimum. If we proceed in the direction of increasing X with constant steps until Y begins to increase,

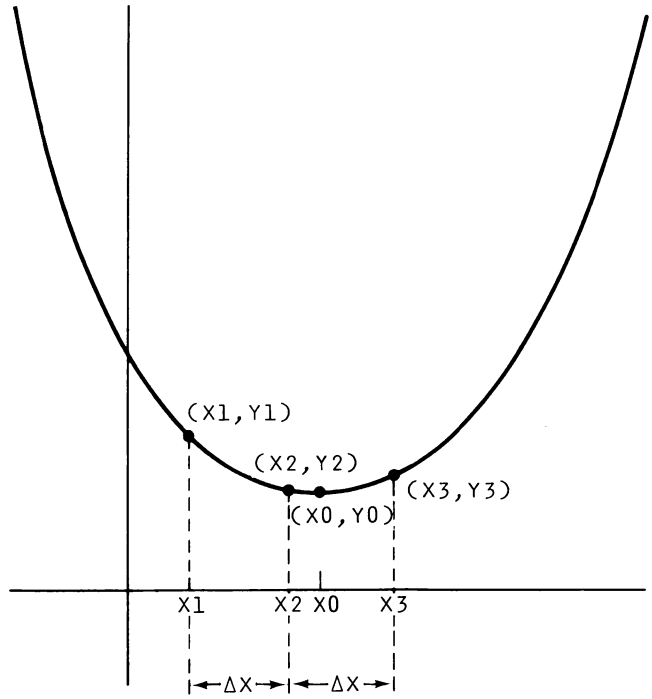


Figure III-11. Minimum of parabola ($X0$, $Y0$) from three points on the parabola

then we can label the last three points ($X1$,$Y1$), ($X2$,$Y2$), and ($X3$,$Y3$), as shown in the figure, and the minimum ($X0$,$Y0$) is a relatively simple function of these points. If we substitute the three known points

$$\begin{aligned} X1 &= X3 - (2 \cdot \Delta X), & Y1 \\ X2 &= X3 - \Delta X, & Y2 \\ & \quad X3, & Y3 \end{aligned}$$

for X and Y in the preceding equation, we get three equations in the three unknowns ($X0$, $Y0$, P). Eliminating $Y0$ and P from these equations by substitution yields

$$X0 = X3 - \frac{\Delta X}{2}\left[\frac{Y1 - (4 \cdot Y2) + (3 \cdot Y3)}{Y1 - (2 \cdot Y2) + Y3}\right]$$

as the location of the minimum. With some algebraic manipulations, this can be written as

$$X0 = X3 - \Delta X\left[\frac{1}{1 + (Y1 - Y2)/(Y3 - Y2)} + \tfrac{1}{2}\right]$$

which minimizes the number of operations required for evaluation.

Refer now to Fig. III-12 to see how the theory can be implemented. The programmer must supply the number of data points (N), the number of adjustable constants in the functional form ($N2$), the starting values for these adjustable constants [$A(I)$, I = 1 to $N2$], and the starting values for the steps away from the starting values [$D(I)$, I = 1 to $N2$]. He

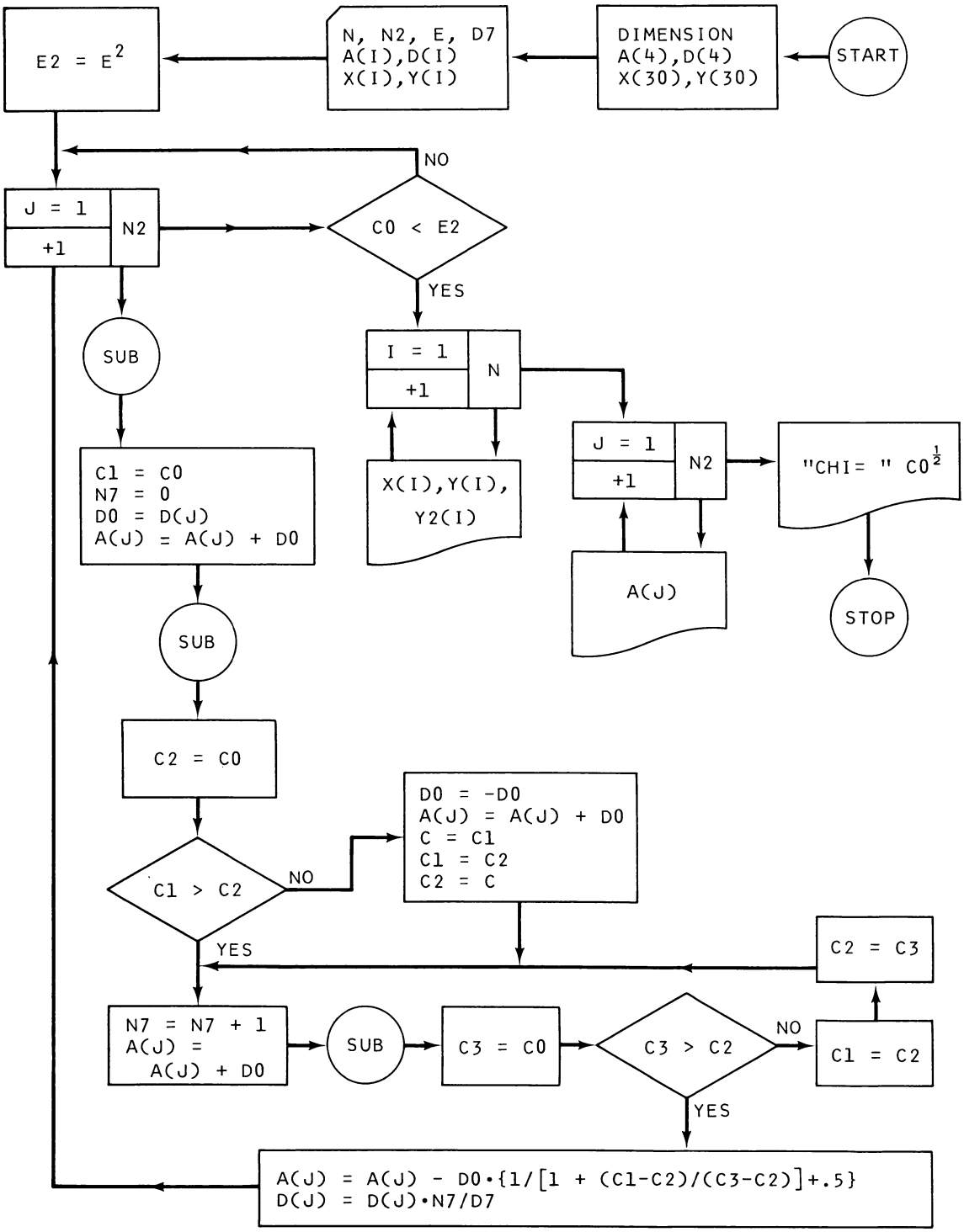


Figure III-12. Non-linear least squares fit (Part 1)

also must supply an allowable maximum (E) for an average deviation of the fitted curve from the data (χ) and the data itself [$X(I)$, $Y(I)$, $I = 1$ to N]. Finally, he supplies a value for $D7$, a constant that determines how much the $D(I)$ are adjusted for the next minimization.

Initial values for the adjustable constants can be obtained from a guess or sometimes from a fit to an equal number of data points. (The rate of convergence can be quite sensitive to the quality of the initial guess.) The step changes in these adjustable constants should be about one-tenth of the constants.

E should be chosen based on the programmer's confidence in his data; if it is accurate to ±.01 and the function is the proper one, then the minimum value for E should be about .01. It is better to start with a larger value, though, because of the very slow convergence of the program with some functions and some data.

If it takes many steps to get to the minimum, then the program automatically increases the size of the step; when it gets close, it reduces the size of the step in order to get a finer value for the location of the minimum. This is accomplished in the statement

```
D(J) = D(J)•N7/D7
```

where $N7$ is a measure of the number of steps. Try using a value of $D7$ of between about 2 and 5, although sometimes larger values may be useful.

After the input variables have been supplied, E^2 is calculated for use as the goodness-of-fit criterion because it is quicker to calculate the average square of the deviations than the average deviation itself.

Then the program calculates the average square deviation for the starting $A(J)$ from the subroutine (Fig. III-13). This is labeled with the variable $C1$ and compared with the average square deviation after incrementing the first adjustable constant, $C2$. If $C2$ is greater than $C1$, then we are going the wrong way, and so the direction of search is reversed ($D0 = -D0$) and $C1$ and $C2$ are interchanged. The next incrementation gives us $C3$, and these three

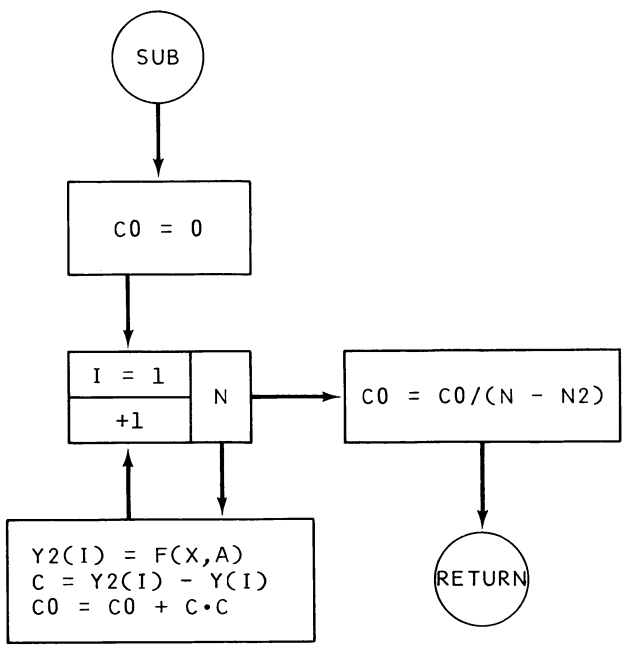


Figure III-13. Non-linear least squares fit (Part 2)

points move along with the incrementation until $C3$ is greater than $C2$, in which case we must have passed the minimum, and the interpolation formula derived earlier is used to locate the minimum.

This large loop for all the adjustable constants may have to be repeated many times. When and if the fit is good enough, the data (X, Y) and the corresponding point $(Y2)$ on the fitted curve are printed, as are the derived constants $[A(I)]$ and χ $(C0^{1/2})$.

The subroutine of Fig. III-13 calculates the average square of the deviations of the data from the fitted curve $\{F(X,A) \equiv F[X(I),A(J)]\}$. In the initial use of your program, you will probably want to print the constants $A(J)$ and χ^2 to observe the tortuous path of minimization.

Problems

1. Function: $F(X) = A(1) \cdot \sin [(4 \cdot X) + A(2)]$
   Starting values: $A(1) = 4, D(1) = .1, A(2) = .05, D(2) = .01$
   Acceptable χ: 1×10^{-4}
   Data: $X(1) = 0$ $Y(1) = .1623$
   $X(2) = .02$ $Y(2) = .5106$
   $X(3) = .04$ $Y(3) = .8557$
   $X(4) = .06$ $Y(4) = 1.195$
   $X(5) = .08$ $Y(5) = 1.527$
   Results: $\chi = 9.75 \times 10^{-5}$, $A(1) = 4.36719$, $A(2) = .0371875$
   [*Note*: The data were generated from $Y = 4.36821 \cdot \sin [(4 \cdot X) + .037159]$ and rounded to four significant figures.]

2. Function: $F(X) = A(1) \cdot \sin \{[A(2) \cdot X] + A(3)\}$
   Starting values: $A(1) = 4, D(1) = .2, A(2) = 1, D(2) = .1, A(3) = .6, D(3) = .1$
   Acceptable χ: .02
   Data:

$X(1) = .3$	$Y(1) = 3.24$	$X(8) = 1.0$	$Y(8) = 3.11$
$X(2) = .4$	$Y(2) = 3.43$	$X(9) = 1.1$	$Y(9) = 2.82$
$X(3) = .5$	$Y(3) = 3.55$	$X(10) = 1.2$	$Y(10) = 2.48$
$X(4) = .6$	$Y(4) = 3.60$	$X(11) = 1.3$	$Y(11) = 2.00$
$X(5) = .7$	$Y(5) = 3.58$	$X(12) = 1.4$	$Y(12) = 1.67$
$X(6) = .8$	$Y(6) = 3.49$	$X(13) = 1.5$	$Y(13) = 1.21$
$X(7) = .9$	$Y(7) = 3.33$		

   Results: $\chi = .0197$, $A(1) = 3.609896$, $A(2) = 1.42831$, $A(3) = .666893$

3. Function: $F(X) = A(1) \cdot \exp [A(2) \cdot X]$ [where $\exp(Y) \equiv e^Y$]
   Starting values: $A(1) = 4$, $D(1) = .2$, $A(2) = .02$, $D(2) = .002$
   Acceptable χ: .01
   Data:

$X(1) = 1$	$Y(1) = 3.77$	$X(6) = 6$	$Y(6) = 4.17$
$X(2) = 2$	$Y(2) = 3.85$	$X(7) = 7$	$Y(7) = 4.26$
$X(3) = 3$	$Y(3) = 3.93$	$X(8) = 8$	$Y(8) = 4.34$
$X(4) = 4$	$Y(4) = 4.01$	$X(9) = 9$	$Y(9) = 4.43$
$X(5) = 5$	$Y(5) = 4.09$		

   Results: $\chi = 6.91 \times 10^{-3}$, $A(1) = 3.69967$, $A(2) = .0200238$

4. Function: $F(X) = \exp \{[A(1) \cdot X] + A(2)\}$
   Starting values: $A(1) = 1$, $D(1) = .2$, $A(2) = .08$, $D(2) = .01$

Acceptable χ: .01
Data: $X(1) = .01$ $Y(1) = 1.095$ $X(6) = .06$ $Y(6) = 1.22$
$X(2) = .02$ $Y(2) = 1.12$ $X(7) = .07$ $Y(7) = 1.24$
$X(3) = .03$ $Y(3) = 1.14$ $X(8) = .08$ $Y(8) = 1.27$
$X(4) = .04$ $Y(4) = 1.17$ $X(9) = .09$ $Y(9) = 1.295$
$X(5) = .05$ $Y(5) = 1.19$ $X(10) = .1$ $Y(10) = 1.32$
Results: $\chi = 5.54 \times 10^{-3}$, $A(1) = 1.9766$, $A(2) = .0771513$

5. Function. $F(X) = A(1) + A(2) \cdot \sin [A(3) \cdot X]$
Starting values: $A(1) = .7$, $D(1) = .07$, $A(2) = 3$, $D(2) = .3$, $A(3) = 1.2$, $D(3) = .1$
Acceptable χ: .02
Data: $X(1) = .1$ $Y(1) = .993$ $X(6) = .6$ $Y(6) = 2.45$
$X(2) = .2$ $Y(2) = 1.31$ $X(7) = .7$ $Y(7) = 2.68$
$X(3) = .3$ $Y(3) = 1.62$ $X(8) = .8$ $Y(8) = 2.88$
$X(4) = .4$ $Y(4) = 1.92$ $X(9) = .9$ $Y(9) = 3.05$
$X(5) = .5$ $Y(5) = 2.19$ $X(10) = 1.0$ $Y(10) = 3.19$
Results: $\chi = .0192$, $A(1) = .615506$, $A(2) = 2.69195$, $A(3) = 1.258$

6. Function: $F(X) = A(1) \cdot \exp [A(2) \cdot X^2]$
Starting values: $A(1) = 10$, $D(1) = .1$, $A(2) = .04$, $D(2) = .004$
Acceptable χ: .01
Data: $X(1) = 1.0$ $Y(1) = 10.23$ $X(6) = 1.5$ $Y(6) = 10.80$
$X(2) = 1.1$ $Y(2) = 10.32$ $X(7) = 1.6$ $Y(7) = 10.94$
$X(3) = 1.2$ $Y(3) = 10.43$ $X(8) = 1.7$ $Y(8) = 11.10$
$X(4) = 1.3$ $Y(4) = 10.54$ $X(9) = 1.8$ $Y(9) = 11.26$
$X(5) = 1.4$ $Y(5) = 10.65$ $X(10) = 1.9$ $Y(10) = 11.45$
Results: $\chi = 9.64 \times 10^{-3}$, $A(1) = 9.81615$, $A(2) = .0423549$

E. Numerical Integration

Even the most complex (continuous) functions in science and engineering can be differentiated exactly, but many differential equations cannot be integrated in closed form. Therefore, in the search for increased programming proficiency with practical application, we turn now to the numerical solution of differential equations. This and the following three sections deal with differential equations of increasing complexity.

It is particularly easy to solve a differential equation of the form

$$\frac{dZ}{dT} = A \cdot F(T)$$

where A is a numerical constant and $F(T)$ is some function of only the variable T. A trivial example of this type of differential equation is the following.

$$\frac{dZ}{dT} = 4.3 \sin (T)$$

(Incidentally, I'm using Z and T as the variable tags rather than the more common Y and X because it makes some later applications easier.)

The general equation has the formal solution, in the region from $T = T1$ to TN of

$$Z = A \int_{T1}^{TN} F(T)\, dT$$

and Z can be interpreted geometrically as the *area* under the $A \cdot F(T)$ curve in the interval from $T1$ to TN (see Fig. III-14).

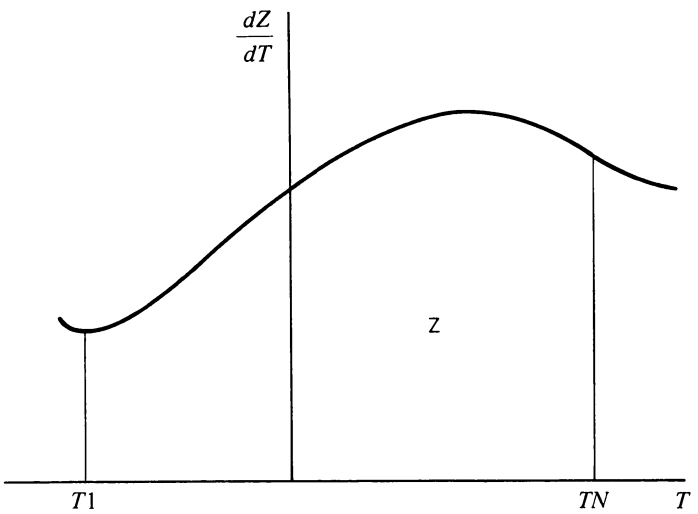


Figure III-14. Geometrical representation of integral,

$$Z = A \int_{T1}^{TN} F(T)dT$$

The approach of numerical analysis is to divide the interval into N very small but equal intervals, find an appropriate value for each of the small areas, and then add them together. The width of each interval is given by

$$\Delta T = \frac{TN - T1}{N}$$

Figure III-15 shows the area of Fig. III-14 divided into only ten small areas for purposes of illustration.

The simplest scheme for calculating the area of the small increments is to assume a straight-line variation of the function between the end points of intervals. Of course this won't be exactly accurate for any function except a straight line, but in theory it can be made arbitrarily accurate by making the interval size sufficiently small (i.e., N sufficiently large). Within this approximation, the area of the first strip is obtained from the average value for dZ/dT multiplied by the interval size

$$\frac{1}{2}\left[\left(\frac{dZ}{dT}\right)_{T1} + \left(\frac{dZ}{dT}\right)_{T2}\right] \Delta T$$

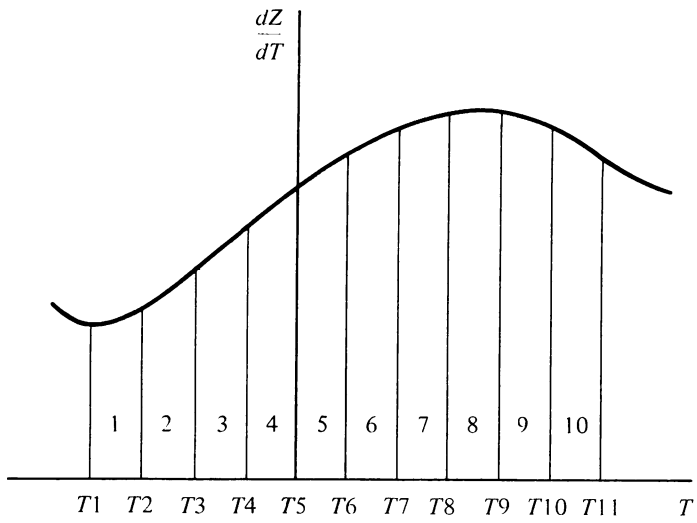


Figure III-15. Numerical evaluation of integral

[where the notation $(dZ/dT)_{T1}$ means the value of the derivative evaluated at $T = T1$]. Reasonably enough, this is called the *trapezoidal rule.*

The next more sophisticated approach, and the one that is used here, is the popular Simpson's Rule. This calculates the approximate area of each strip by fitting a second-order polynomial (the parabola of Sec. D) to the function there. The result of such an analysis is that the total area, and thus the integral, is given approximately by

$$Z \simeq A \frac{\Delta T}{3} \left[\left(\frac{dZ}{dT}\right)_{T1} + 4\left(\frac{dZ}{dT}\right)_{T2} + 2\left(\frac{dZ}{dT}\right)_{T3} + 4\left(\frac{dZ}{dT}\right)_{T4} + \ldots + 2\left(\frac{dZ}{dT}\right)_{TM-2} + 4\left(\frac{dZ}{dT}\right)_{TM-1} + \left(\frac{dZ}{dT}\right)_{TM}\right]$$

where M (the number of interval end points) is equal to $N + 1$ and *must* be an *odd* integer equal to or greater than 5. In Fig. III-15, M is equal to 11, and the total area Z is given approximately by

$$Z \simeq A \frac{\Delta T}{3} \left[\left(\frac{dZ}{dT}\right)_{T1} + 4\left(\frac{dZ}{dT}\right)_{T2} + 2\left(\frac{dZ}{dT}\right)_{T3} + 4\left(\frac{dZ}{dT}\right)_{T4} + 2\left(\frac{dZ}{dT}\right)_{T5} + 4\left(\frac{dZ}{dT}\right)_{T6} + 2\left(\frac{dZ}{dT}\right)_{T7} + 4\left(\frac{dZ}{dT}\right)_{T8} + 2\left(\frac{dZ}{dT}\right)_{T9} + 4\left(\frac{dZ}{dT}\right)_{T10} + \left(\frac{dZ}{dT}\right)_{T11}\right]$$

[Simpson's Rule is usually given with any multiplying constant A contained within the definition of $F(T)$, but I have factored it out to emphasize that in the interests of efficiency, the multiplication should be made after the small areas have been evaluated and added.]

Similarly, we should evaluate and add all the small areas being multiplied by 2 or 4 *before* doing the multiplying. Thus the calculation can be symbolized by the following grouping of like terms:

$$Z \simeq A \frac{\Delta T}{3} \left\{\left[\left(\frac{dZ}{dT}\right)_{T1} + 4\left(\frac{dZ}{dT}\right)_{T2} + \left(\frac{dZ}{dT}\right)_{T11}\right] + 2\left[\left(\frac{dZ}{dT}\right)_{T3} + \left(\frac{dZ}{dT}\right)_{T5} + \left(\frac{dZ}{dT}\right)_{T7} + \left(\frac{dZ}{dT}\right)_{T9}\right] + 4\left[\left(\frac{dZ}{dT}\right)_{T4} + \left(\frac{dZ}{dT}\right)_{T6} + \left(\frac{dZ}{dT}\right)_{T8} + \left(\frac{dZ}{dT}\right)_{T10}\right]\right\}$$

This grouping puts the second term with the first and last so that equal numbers of terms are in the other two groups. The flowchart in Fig. III-16 details the logic required to program the calculation in this fashion. (You might have been tempted to use ΔT as the incrementing variable in the loop—valid in some computer languages—but using the integer increment prevents an accumulation of round-off error.)

The next move is yours. Write a program from the flowchart. It is suggested that you make it easy to run the program a second time with a different N. so that you can easily see the effect of reducing the size of the interval. Recall too that N must be an even integer equal to or greater than 5.

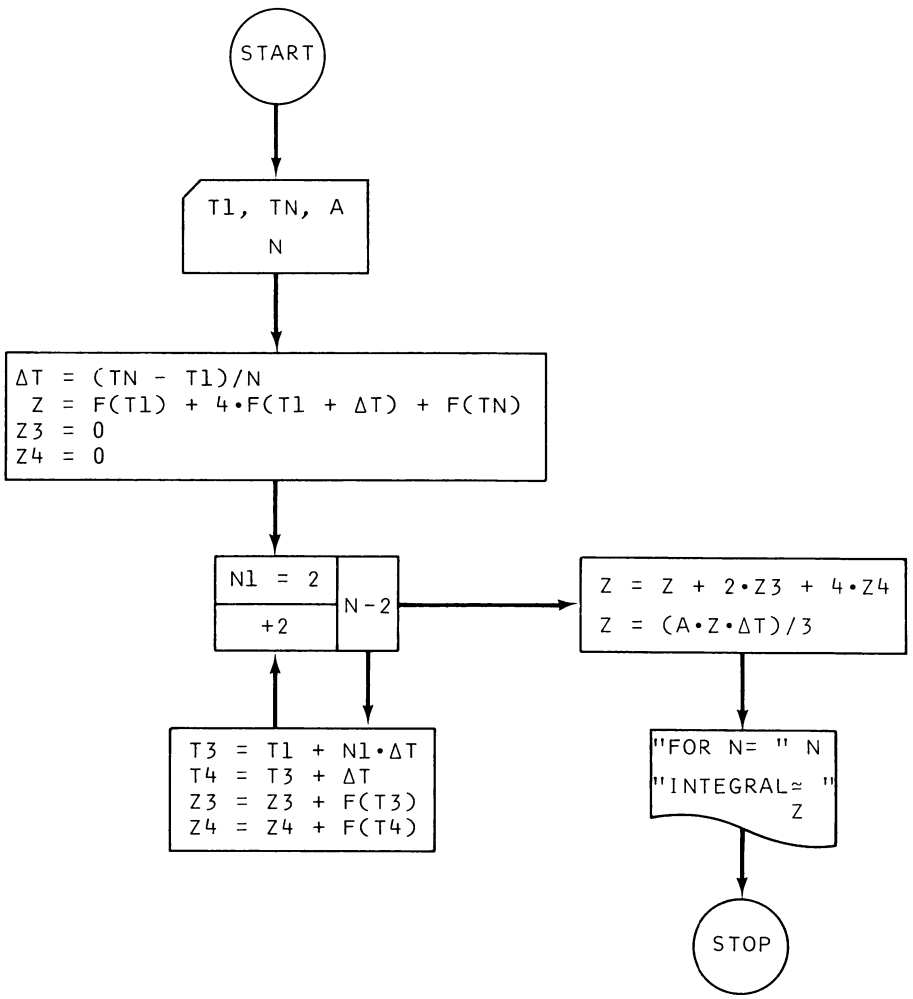


Figure III-16. Numerical integration by Simpson's Rule

Do you realize why it is better to use Simpson's Rule than to use the trapezoidal approximation? Cannot we obtain the same accuracy with a trapezoidal-rule program by just reducing the size of ΔT? The answer is *yes* in theory but usually *no* in practice. The explanation is that the accumulated round-off error increases with the increasing number of terms as ΔT is decreased, so that at a certain point the accuracy is *degraded* by decreasing ΔT. I'll demonstrate that in a moment.

How are you going to check out your program? The customary technique, and one that is very effective here, is to try some questions for which you know the answer. (This intimidates the computer by showing it that you're not dumb.) We can start with $F(T)$ as a second-degree polynomial, because we would expect Simpson's Rule to be particularly good for this case, considering its derivation.

$$4\int_{-2}^{4} T^2\,dT = \left[\frac{4T^3}{3}\right]_{-2}^{4} = \frac{4}{3}\left[4^3 - (-2)^3\right] = 96$$

For this integral, my program yields exactly 96 for N equal to 10, 100, 1000, and 10000, but the last run required almost a minute for execution. Thus the numerical integration scheme does work very well with a second-degree polynomial function. But notice how costly in time a very small interval is!

Moving on to a more challenging integral,

$$\int_{-2}^{4} e^{2T}\,[\sin(3T)]\,dT = e^{2T}\left[[2\sin(3T) - 3\cos(3T)/13\right]_{-2}^{4} = -826.572$$

and the program yields $N = 10$, Integral $= -718.288$; $N = 100$, Integral $= -826.565$; $N = 1000$, Integral $= -826.572$; $N = 10000$, Integral $= -826.573$ (4 minutes). Did you see that? The answer for $N = 1000$ is more accurate than for $N = 10000$!

Problems: Evaluate the following integrals.

1. $5\int_{0}^{\pi/2} \sin(x) \cdot \cos^4(x)\,dx$ *Answer*: 1

2. $\int_{-8}^{2} \sinh^2(x)\,dx$, where $\sinh(x) \equiv \left[\frac{e^x - e^{-x}}{2}\right]$ *Answer*: 1.17649E+06

3. $\int_{0}^{.5} [\arcsin(x)]^2\,dx$, where $\arcsin(x) \equiv \arctan\left[\frac{x}{(1-x^2)^{1/2}}\right]$ *Answer*: .0439775

4. $\int_{1}^{2.5} \frac{dx}{x^2(7.68 - x^2)}$ *Answer*: .27425

F. Numerical Solution of First-Order Differential Equations

Section E presented a special case; the more general form for an ordinary differential equation of the first order and first degree is

$$\frac{dZ}{dT} = F(T, Z)$$

We can't plot the derivative as a function of T, as we did in Sec. E, because it depends on both T and Z itself. However, suppose that the actual functional variation of Z is that given by the curve in Fig. III-17. In problems of physical interest, we usually have boundary conditions that tell us the value of T and Z at a particular point (call them $T1$ and $Z1$), and we wish to determine the value of Z (call it ZN) at a different point (TN).

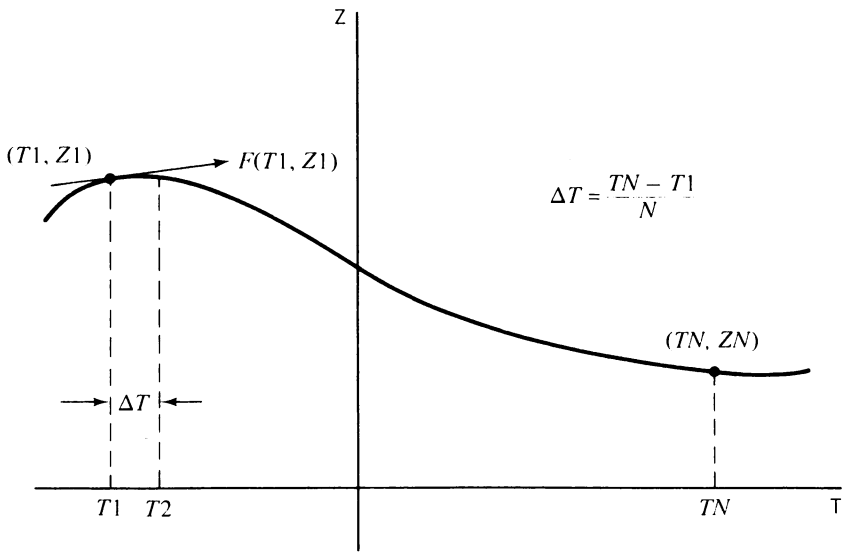


Figure III-17. Representation of solution to first-order differential equation

The simplest solution technique is to move a small horizontal distance ΔT along the line representing the slope at point $T1$, call this new point $T2$, evaluate the slope there, and move another horizontal distance ΔT along this new slope line, etc. Obviously, unless the curve is a straight line, error will be introduced at each new point, and this error will *accumulate*. A better technique is to somehow estimate the *average slope* between adjacent points. The Runge-Kutta fourth-order method uses this approach and is perhaps the most generally useful of all such techniques. The slope it uses is a weighted average of slopes computed at the end points of the interval *and* an intermediate point.

Refer now to Fig. III-18 for a flowchart of this method. The programmer supplies the integration interval ($T1$ to TN), the value of Z at the beginning of the interval ($Z1$), and the number of divisions of the interval (N). Generally you should make N equal to about 100, because this usually gives an answer correct to about six significant figures; a larger N often

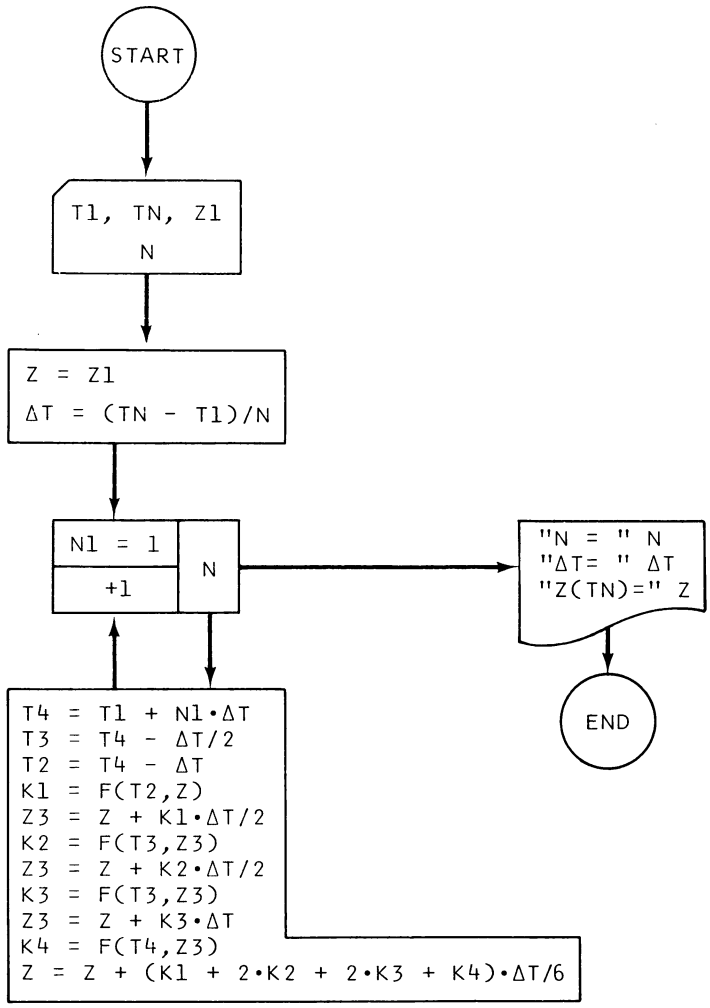


Figure III-18. Numerical solution of first-order differential equation

requires rather long execution times (a minute or more). As in Sec. E, the computed value changes rapidly as N is initially increased, and then, after the correct value is reached, it tends to change quite slowly due to round-off error with further increases in N.

In the loop the values for the end points are based on the starting value ($T1$) so that no round-off error can accumulate in their values. Thus $T2$ is the beginning of the interval being evaluated, $T3$ is the midpoint, and $T4$ is the end point.

$K1$ is the slope at $T2$; an estimate is made of the value of Z at the midpoint ($Z3$) by using this initial slope. With this value of Z at the midpoint, an estimate can be made for the slope there ($K2$). This slope is used to make a separate estimate for Z at the midpoint, and this leads to another estimate for the midpoint slope ($K3$). Finally, the end point Z is estimated from this $K3$ slope, and this is used to obtain the estimated slope at the end point ($K4$).

Note that if the function were a straight line so that all the calculated slopes were the same ($K = K1 = K2 = K3 = K4$), then the equation using weighted values for the slopes would clearly give the correct answer [$Z = Z + (K \cdot \Delta T)$], and this gives us some confidence in the formula.

You can make an initial check of your program using the functions and answers of Sec. E (with $Z1$ equal to any arbitrary value, because there is no functional dependence on Z). A very useful debugging trick is to print the values of $K1$, $K2$, $K3$, and $K4$ in the loop for a few iterations; if they are being correctly calculated, then $K2$ and $K3$ should be about halfway between $K1$ and $K4$.

Problems: Solve the following differential equations for the boundary conditions and end point that are indicated.

1. $\frac{dZ}{dT} = [\exp(T^2)] \cdot \cos(T) - (T \cdot Z^2)$ $T1 = 0, TN = 1, Z1 = 1$ *Answer:* 1.30375

2. $\frac{dZ}{dT} = (-T \cdot e^{-T}) + (T \cdot Z)$ $T1 = 1, TN = 2, Z1 = 3$ *Answer:* 12.5852

3. $\frac{dY}{dX} = X^2 - [Y^2 \cdot \sin(X)]$ $X1 = -1, XN = 3, Y1 = 0$ *Answer:* 4.43083

4. $\frac{dR}{d\theta} = (\theta^2 \cdot e^{-R}) + [3 \cdot \cos(\theta)]$ $\theta 1 = -1, \theta N = 1, R1 = 1$ *Answer:* 6.12698

G. Numerical Solution of Second-Order Differential Equations

Why stop with *first*-order differential equations? Second-order differential equations aren't that much more difficult to solve, and they abound in the mathematical description of nature (deflection equations, circuit theory, Newton's second law, Maxwell's equations, the Schroedinger equation, etc.). Once again we limit ourselves to first-degree differential equations (those for which the derivatives are not squared, cubed, etc.).

The general form of a second-order first-degree differential equation is

$$\frac{d^2Z}{dT^2} = F\left(T, Z, \frac{dZ}{dT}\right)$$

If we use the symbol Q to represent the first derivative of Z with respect to T, then we can write this equation as two simultaneous equations:

$$\frac{dZ}{dT} \equiv Q$$

$$\frac{dQ}{dT} = F(T, Z, Q)$$

The Runge-Kutta method solves these together, inching along the T axis as before. We assume that the boundary conditions for the physical problem give us an initial value ($Z1$) for Z *and* an initial value for its first derivative ($Q1$) at a point $T1$ and that the solution is desired at some distant point TN.

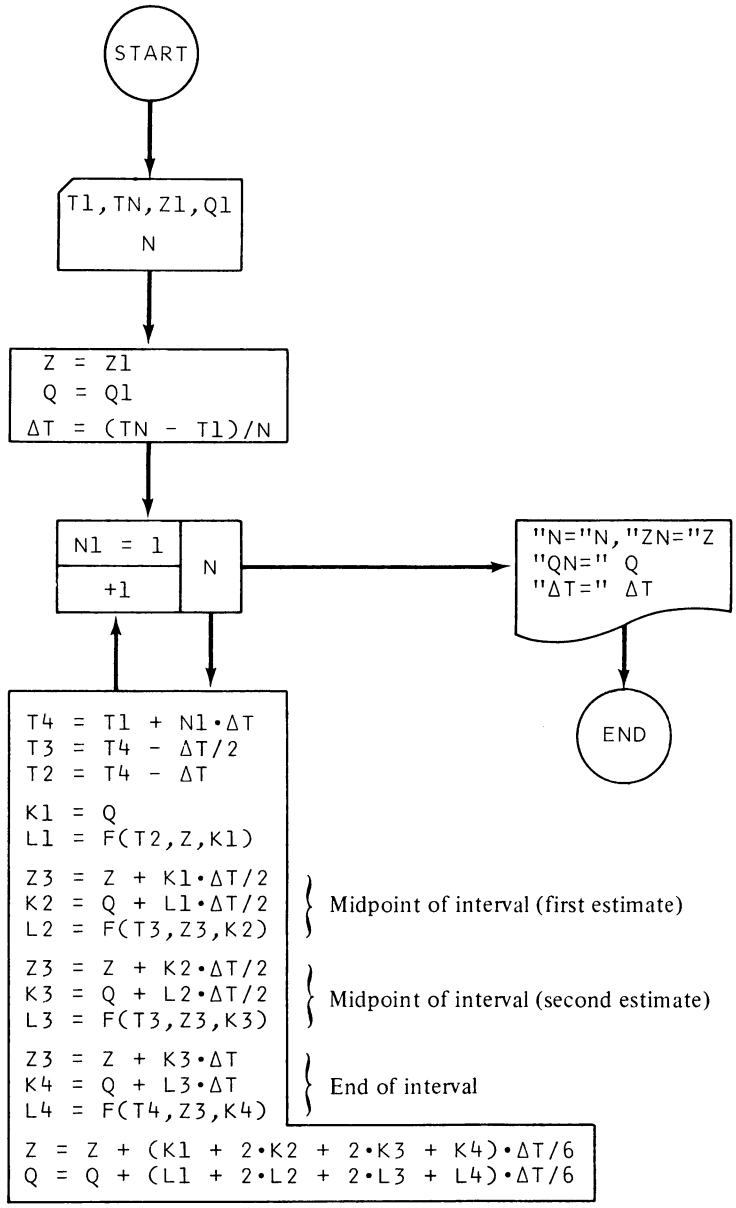


Figure III-19. Numerical solution of second order differential equation

Refer now to Fig. III-19 for the flowchart for this method. After the programmer supplies the boundary conditions, the actual solution takes place in the loop, as before. First the initial value of dZ/dT ($Q1$) is used to estimate the value of Z at the midpoint of the interval ($Z3$). Then, based on the rate of change in dZ/dT at the *beginning* of the interval ($L1$), the value of dZ/dT at the midpoint of the interval ($K2$) is estimated. This value for dZ/dT is used to estimate d^2Z/dT^2 at the midpoint ($L2$). A new estimate for Z at the midpoint ($Z3$ again) gives a new estimate for dZ/dT and d^2Z/dT^2 there ($K3$ and $L3$, respectively). The last bootstrap uses these values to obtain values for Z, dZ/dT, and d^2Z/dT^2 at the end of the interval. Finally, the solution for both Z and dZ/dT at the end of the interval is obtained from the weighted average of their derivatives, as before.

Debugging your program will again be greatly facilitated by watching the K's and L's in the loop; they should change gradually and progressively from their initial values (which you can easily determine).

Problems: Solve the following differential equations for the boundary conditions and end point that are indicated.

1. $\frac{d^2Z}{dT^2} = 3 \cdot Z - \frac{dZ}{dT} + e^T$ $\quad T1 = 0, TN = 1, Z1 = 1, Q1 = 0$
   Answers: $ZN = 3.04675, QN = 4.63174$
2. $\frac{d^2Z}{dT^2} = -3 \cdot \frac{dZ}{dT} - Z + e^{-2 \cdot T}$ $\quad T1 = -1, TN = 0, Z1 = 0, Q1 = 1$
   Answers: $ZN = 1.05653, QN = .415296$
3. $\frac{d^2Y}{dX^2} = -\frac{dY}{dX} - 2 \cdot Y + \sin(X)$ $\quad X1 = -2, XN = 2, Y1 = 2, Q1 = 2$
   Answers: $YN = .504949, QN = .942506$
4. $\frac{d^2R}{d\theta^2} = e^{\theta} \cdot \sin(2 \cdot \theta) - 3 \cdot \frac{dR}{d\theta} + 2 \cdot R$ $\quad \theta 1 = 0, \theta N = 1, R1 = 5, Q1 = 5$
   Answers: $RN = 9.9477, QN = 6.19639$

H. Coupled Differential Equations

As a final foray into differential equations, we turn our attention to a particular problem that leads us to coupled differential equations.

In an introductory physics course, you are given equations that purport to describe projectile flight. They are derived by ignoring air drag and by then separately solving the equations of motion in the vertical and horizontal directions. Physically, being able to solve them separately means that the motions are independent of each other; what happens in the horizontal direction is independent of what happens in the vertical direction (until the projectile hits the ground).

But for almost all common projectiles, air drag is extremely important. The projectile path is *not* a parabolic curve but has greater curvature on the way down. The speed upon return to the earth is *not* the speed with which it left the earth. (Do you think outfielders would object to catching baseballs just as they come off the bat?)

This air drag is best described mathematically for most common projectiles as a backward force proportional to the *square* of the projectile's speed. In the horizontal direction, the air drag is therefore always in the same direction, but vertically, air drag is a *downward* force on the way up and an *upward* force on the way down; it aids gravity in limiting the maximum vertical height attained but then opposes gravity in the buildup of speed on the way down.

In Fig. III-20 the flight path of a projectile leaving the ground at a speed V_0 and at an angle θ_0 to the horizontal is depicted. Also shown are force diagrams for the upward part and the downward part. The symbol k represents a drag constant such that the air drag force is given by kV^2, where V is the instantaneous velocity (the derivative of position with respect to time).

The motion is occurring in two dimensions, so it requires two equations for its mathematical description. We choose the horizontal direction to be the X direction and the vertical direction to be the Y direction; then the velocity components are dX/dT and DY/dT, as shown in Fig. III-21.

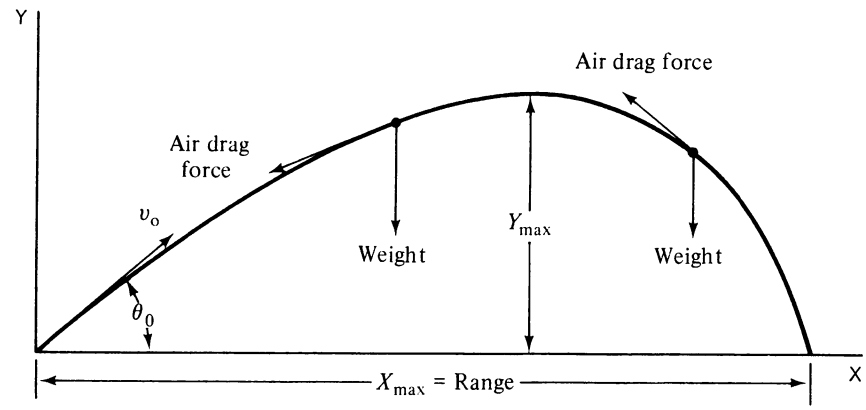


Figure III-20 Flight path of a projectile with air drag

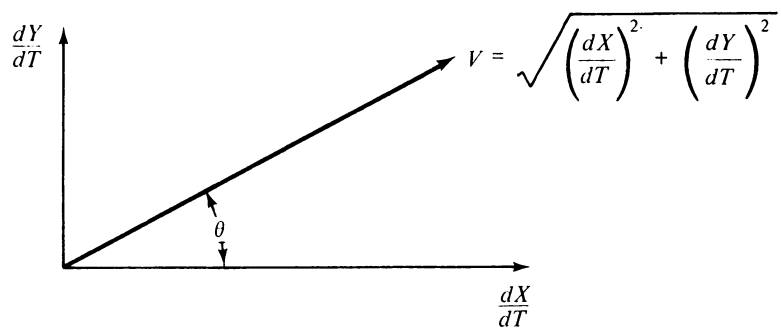


Figure III-21. The projectile's instantaneous velocity and its velocity components

From this figure it can be seen that

$$V = \sqrt{\left(\frac{dX}{dT}\right)^2 + \left(\frac{dY}{dT}\right)^2} \tag{III-1}$$

$$\cos(\theta) = \frac{\frac{dX}{dT}}{V} \tag{III-2}$$

$$\sin(\theta) = \frac{\frac{dY}{dT}}{V} \tag{III-3}$$

in which θ is the instantaneous angle of the flight direction relative to the horizon.

Newton's Second Law states that the net force on an object in any direction is equal to the mass of that object multiplied by its acceleration in that direction (the *second* derivative

of position with respect to time). Thus we obtain

$$-kV^2 \cos(\theta) = m \frac{d^2X}{dT^2} \tag{III-4}$$

$$-kV^2 \sin(\theta) - mg = m \frac{d^2Y}{dT^2} \tag{III-5}$$

Ascending

and

$$-kV^2 \cos(\theta) = m \frac{d^2X}{dT^2} \tag{III-6}$$

$$+kV^2 \sin(\theta) - mg = m \frac{d^2Y}{dT^2} \tag{III-7}$$

Descending

in which m is the mass, mg is the weight, and g is the acceleration due to gravity, which can be assumed constant and equal to 32.2 ft/sec^2. To simplify these equations, we define a new drag constant, $c \equiv k/m$, and substitute for V, $\cos(\theta)$, and $\sin(\theta)$ from Eqs. III-1, III-2, and III-3. The result is

$$\frac{d^2X}{dT^2} = -c \frac{dX}{dT} \sqrt{\left(\frac{dX}{dT}\right)^2 + \left(\frac{dY}{dT}\right)^2} \tag{III-8}$$

$$\frac{d^2Y}{dT^2} = -c \frac{dY}{dT} \sqrt{\left(\frac{dX}{dT}\right)^2 + \left(\frac{dY}{dT}\right)^2} - g \tag{III-9}$$

Ascending

$$\frac{d^2X}{dT^2} = -c \frac{dX}{dT} \sqrt{\left(\frac{dX}{dT}\right)^2 + \left(\frac{dY}{dT}\right)^2} \tag{III-10}$$

$$\frac{d^2Y}{dT^2} = +c \frac{dY}{dT} \sqrt{\left(\frac{dX}{dT}\right)^2 + \left(\frac{dY}{dT}\right)^2} - g \tag{III-11}$$

Descending

Now you can observe the coupling explicitly; dY/dT appears in the equation for d^2X/dT^2, and dX/dT appears in the equation for d^2Y/dT^2. Somehow we must solve them together.

A procedure that works well in this case is to make a *guess* of the value of dY/dT at the end of the next time interval, use this guess to solve Eq. III-8 or III-10 for dX/dT at the end of this same time interval, and use this dX/dT to obtain the corresponding *calculated* dY/dT from Eq. III-9 or III-11. This calculated dY/dT for the end of the next time interval generally is not the same as the guessed value, and somehow a correction must be made to make them agree.

We can think of the quantity $[(dY/dT)_{\text{Calc}} - (dY/dT)_{\text{Guess}}]$ as being a function of $(dY/dT)_{\text{Guess}}$, and then we are equivalently looking for the zero crossing of this function so that a root-finding technique can be used. In this case, though, an iteration method is simple and converges quite rapidly; with this method, one uses $(dY/dT)_{\text{Calc}}$ as the next guess and continues with this process until $[(dY/dT)_{\text{Calc}} - (dY/dT)_{\text{Guess}}]$ is satisfactorily small.

Our first guess doesn't have to be a blind one, though. For the very first time increment ΔT, we can neglect the air drag term and guess that dY/dT at $T = 0 + \Delta T$ is its initial value, $V_0 \sin(\theta_0)$, minus $g \cdot \Delta T$. After that we can use the actual change in dY/dT in the previous time interval to guess the new dY/dT.

There are some changes to be made in our Runge-Kutta program—like rewriting it. The automatic loop has to go; we can't actually increment the variables until satisfactory convergence of the two equations has been attained. Also, there is no need to calculate a new X or Y until after convergence, because none of the equations contain these variables directly (because what happens depends on the speeds but not on the position relative to the ground).

When you obtain a tentative value for dX/dT at the end of the time interval from Eq. III-8, you use it in solving Eq. III-9 or III-11 for dY/dT at the end of the same time interval. But the Runge-Kutta method requires values at the beginning and midpoint as well as at the end point. The beginning values you know for both dX/dT and dY/dT; you should make your midpoint guess for dX/dT in Eq. III-9 or III-11 equal to the *average* of its known beginning value and the guess of its end point value.

Assuming that we wish to follow the projectile all the way to the ground, we must make a check to see when dY/dT first becomes zero or negative; that is when we're at the top of the motion and must switch from Eqs. III-8 and III-9 to Eqs. III-10 and III-11. Finally we should stop when Y first becomes equal to or slightly less than zero, because we've just hit the ground. We want to determine the location of the maximum height, the maximum horizontal distance (the range), the time of flight to the maximum height, and the total time of flight.

In Fig. III-22 I've charted the general approach just described in words. The details are missing from the chart this time, and drawing your own detailed flowchart and being sure you understand it may be the fastest way to solve the problem.

For a check on your program, try to reproduce the record major league home run: 565 feet. To obtain this range, use a drag constant c equal to .00265 (about right for a baseball for distances in feet and times in seconds), and for the initial values use

V_0 = 319.7 ft/sec (equal to 218 miles per hour)
θ_0 = 32° (remember to convert to radians!)
E = 1×10^{-5}

Other results of the calculation are

T (highest point) = 2.59 seconds
T (total flight time) = 5.38 seconds
Y (max) = 158 feet
V (impact) = 141 ft/sec (equal to 96 miles per hour)

As a preliminary check on your program, check your answers with the exact values for no drag (set c equal to zero):

Range = 2853.5 feet
T (highest point) = 5.26 seconds
T (total flight time) = 10.52 seconds
Y (max) = 445.8 feet

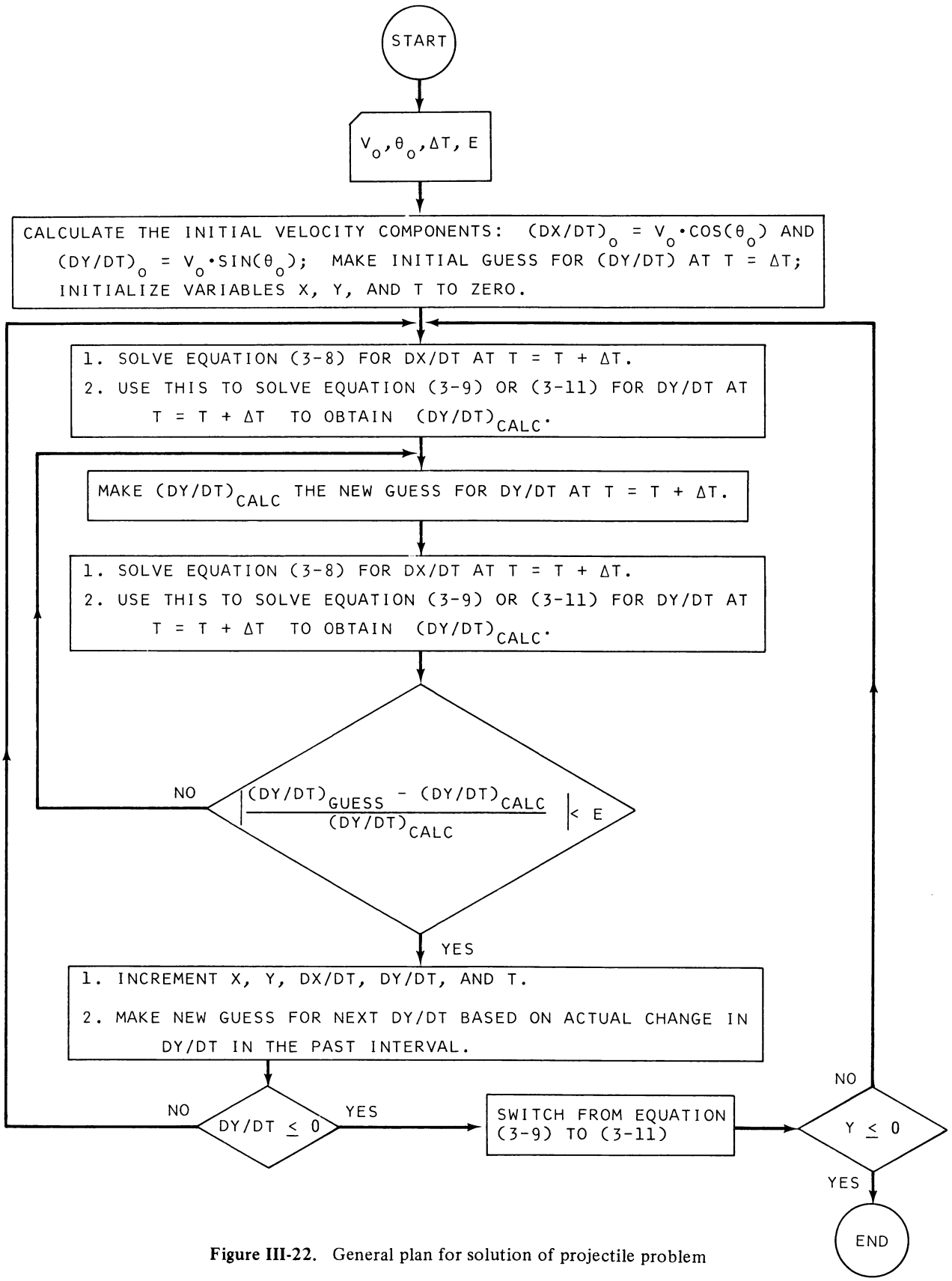


Figure III-22. General plan for solution of projectile problem

You shouldn't need a time interval less than 0.01 second; start with about 0.1 seconds to keep the execution time down during debugging. When you have your program working properly, you can explore the real world of projectiles: See how the maximum range occurs at smaller and smaller values of θ_0 as the drag constant increases, plot the actual shape of the flight path for various drag values, etc. (Fig. III-20 is the shape of the flight path for an initial speed of 160 miles per hour.)

In general you can expect coupled differential equations to appear whenever a physical process described by a differential equation is taking place in two or more dimensions and whenever the activity in one dimension is linked with what happens in the other.

I. Errors and Error Studies

If you've programmed your way to this section, you've surely seen plenty of errors already. Presumably these have been mostly the programming bug type of error, and bugs of this nature can live forever because there is no possible way to check a complex program for every input combination. In scientific programming the usual approach is to test a program with a situation sufficiently simple that one already knows the answer. This is a tremendously valuable tool, but it surely doesn't *prove* that the complex solution is also correct. Sometimes a programmer can develop a feeling for how a calculation should proceed, for example the rate of convergence, and thereby be able to *smell* a bug. In any case, as a scientist or engineer, you have to be sure that you have modeled the physical process properly, and you are usually the one in the best position to devise checks to determine the reasonableness of the answers. (Recall the legendary counsel of the battle-worn veteran programmer, "GIGO," which translates as "GARBAGE IN → GARBAGE OUT.")

In this section we consider other types of errors. One of these other types has already been confronted; it is that resulting from the computing process itself. Examples are the accumulated round-off error in a loop if the incrementation is made inexact through the decimal to binary conversion and the errors that result when large and small numbers are added and subtracted. Usually this type of problem can be exposed by studying the results of a calculation at each intermediate point.

The type of error that most often concerns the physical scientist, though, is error in his data. There may be fluctuations in his measurements simply because he is measuring a statistical property, such as nuclear decay rates, for which cause and effect are no longer individually operative; but our concern here is primarily with those errors that emanate from instrumental limitations. Seldom do our voltmeters give us more than three- or four-digit accuracy, for example. One of the simplest ways to study the propagation of uncertainties is through *worst-case* error analysis.

If we measure a voltage $V = 6.37 \pm .03$ volts and a current $I = 1.23 \pm .05$ amperes for a resistor, we can say that it has a resistance of

$$R \equiv \frac{V}{I} = \frac{6.37}{1.23} = 5.18 \text{ ohms}$$

But with the worst luck the voltage would have its *maximum* possible estimated value while the current would simultaneously have its *minimum* possible value, giving a maximum possible resistance of

$$R = \frac{6.40}{1.20} = 5.33 \text{ ohms}$$

or the voltage would have its *minimum* possible value while the current had its *maximum* estimated value, giving a minimum possible resistance of

$$R = \frac{6.34}{1.26} = 5.03 \text{ ohms}$$

Therefore, we would be completely safe in stating that the resistance is

$$R = 5.18 \pm .15 \text{ ohms}$$

The simplest way to make this kind of analysis on the computer is to loop through all possible combinations of nominal, maximum, and minimum values for each variable and pick out the maximum and minimum values from the printed output. Alternately, for many variables, you can keep track of the value for each variable that gives a maximum or minimum and then use these at the end to get the overall maximum values (but this assumes independence of the variables). Another way, if your errors are small and you have a formula in closed form, is to use differentials.

An appealing alternative is to use the random function generator to do a statistical error analysis study. The most likely distribution of random observations for most experiments is the Gaussian distribution. It is drawn in Fig. III-23 for a measurement of voltage; the distribution shown is appropriate for an average voltage measurement of 4.0 volts with an average (standard) deviation δ of ±.2 volts. The ordinate indicates the relative probability

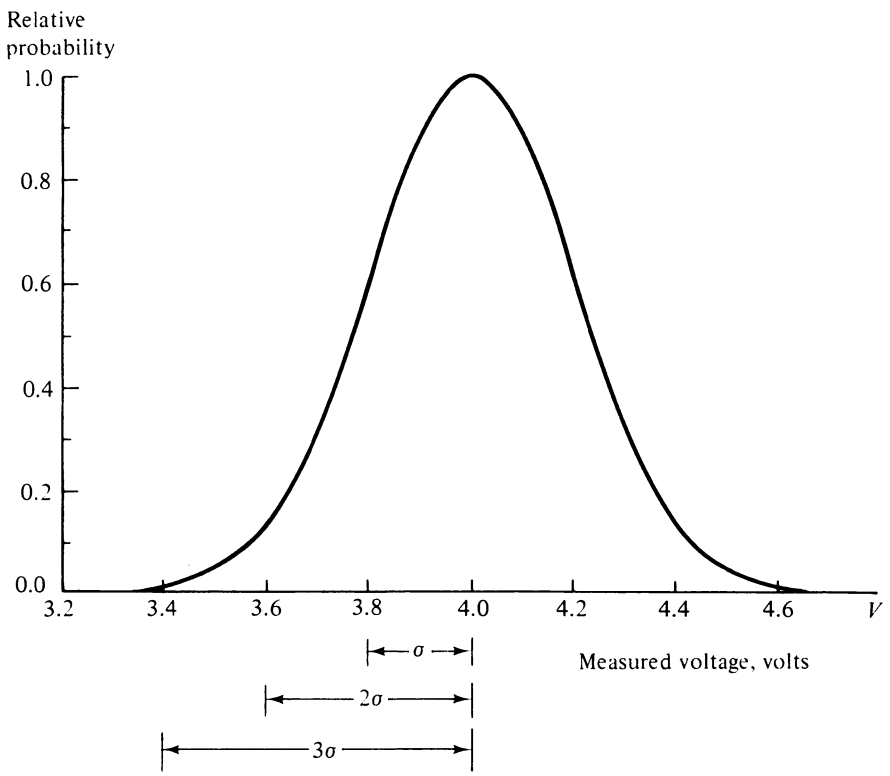


Figure III-23. Gaussian distribution, voltage measurements

of making a particular voltage measurement; thus the probability of making a measurement of just one standard deviation away from the average (3.8 or 4.2 volts) is about 61 percent of the probability for measuring the average value (4.0 volts). By three standard deviations away from the average or mean value (3.4 or 4.6 volts), the probability is only one percent of the probability for the average.

The mathematical expression for the Gaussian distribution, from which Fig. III-23 was obtained, is

$$P = \exp\left[-\frac{1}{2}\left(\frac{V - \overline{V}}{\sigma}\right)^2\right]$$

in which P is the relative probability of measuring a voltage V for an average value of $\overline{V}$ and a standard deviation σ.

These relative probabilities are listed in Table III-1. The right-hand column of this table lists the relative probability multiplied by ten and rounded to the nearest integer; this then is a frequency distribution for a total of 129 + 119 = 248 measurements. (The second number, 119, is the number of measurements of *negative* deviations from the average.)

We can now define a subscripted variable that contains this frequency distribution. An example is the variable $G(I)$ given in Table III-2.

TABLE III-1. GAUSSIAN PROBABILITY AND FREQUENCY DISTRIBUTION

Measured Value	*Relative Probability*	*Frequency Distribution*
Average	1.000	10
Average + .1σ	0.995	10
Average + .2σ	0.980	10
Average + .3σ	0.956	10
Average + .4σ	0.923	9
Average + .5σ	0.882	9
Average + .6σ	0.835	8
Average + .7σ	0.783	8
Average + .8σ	0.726	7
Average + .9σ	0.667	7
Average + 1.0σ	0.607	6
Average + 1.1σ	0.546	5
Average + 1.2σ	0.487	5
Average + 1.3σ	0.430	4
Average + 1.4σ	0.375	4
Average + 1.5σ	0.325	3
Average + 1.6σ	0.278	3
Average + 1.7σ	0.236	2
Average + 1.8σ	0.198	2
Average + 1.9σ	0.164	2
Average + 2.0σ	0.135	1
Average + 2.1σ	0.110	1
Average + 2.2σ	0.089	1
Average + 2.3σ	0.071	1
Average + 2.4σ	0.056	1
Average + 2.5σ	0.044	0
Average + 2.6σ	0.034	0
Average + 2.7σ	0.026	0
Average + 2.8σ	0.020	0
Average + 2.9σ	0.015	0
Average + 3.0σ	0.011	0
	Total	129

TABLE III-2. USING A VARIABLE $G(I)$ TO REPRESENT THE FREQUENCY DISTRIBUTION

$G(I)$ =	I =
0	1 → 10
.1	11 → 20
− .1	21 → 30
.2	31 → 40
− .2	41 → 50
.3	51 → 60
− .3	61 → 70
.4	71 → 79
− .4	80 → 88
.5	89 → 97
− .5	98 → 106
.6	107 → 114
− .6	115 → 122
⋮	⋮
1.7	227 → 228
−1.7	229 → 230
1.8	231 → 232
−1.8	233 → 234
1.9	235 → 236
−1.9	237 → 238
2.0	239
−2.0	240
2.1	241
−2.1	242
2.2	243
−2.2	244
2.3	245
−2.3	246
2.4	247
−2.4	248

Then if we write

```
I = INT(248•RND(X) + 1)
```

where INT is an operator that determines the truncated integer value of what is inside the parenthesis and where RND(X) is a random function generator that returns a random number between 0 and 1, we get a random number between 1 and 248; from these the randomized value of the variable is obtained from

$$Z = Z_0 + \delta \cdot G(I)$$

where Z_0 is the nominal value for the variable. This should be done for each variable in turn, each time using a new random number to obtain the index for $G(I)$ and using the appropriate δ for that variable.

After we have made a sufficient number of simulations and obtained enough calculated answers, we can learn some facts about the distribution of answers without having to print or plot all of them. Specifically, it seems reasonable to ask for the average value of the calculated answers, the average deviation from this average value, and the maximum and minimum values among the answers.

The maximum and minimum values can be found by assuming that the first calculation is both the maximum and minimum value and by replacing these values when succeeding calculations give larger or smaller values respectively. We can obtain the average value easily enough by maintaining a running total and dividing by the number of simulations at the end.

It might seem that the only way to obtain the standard deviation from the average is to store all the calculated values and then calculate the deviation after the average value itself has been determined. This could require a lot of storage space in the computer, and fortunately it isn't necessary. Suppose that we denote the calculated values by Z_i and the average of these values by $\bar{Z}$. Then the standard deviation σ for N calculations is defined as

$$\sigma \equiv \sqrt{\frac{\Sigma(Z_i - \bar{Z})^2}{N}}$$

where the index i takes on integer values from 1 to N.

Suppose then that we maintain a running total for the quantity Z_i^2. At the end of the simulations, after we have calculated $\bar{Z}$, we can determine σ from this sum, as shown by multiplying out the numerator in the defining equations for σ.

$$\begin{aligned}\Sigma(Z_i - \bar{Z})^2 &= \Sigma Z_i^2 - 2 \cdot \bar{Z} \cdot \Sigma Z_i + N \cdot \bar{Z}^2 \\ &= \Sigma Z_i^2 - 2 \cdot N \cdot \bar{Z}^2 + N \cdot \bar{Z}^2 \\ &= \Sigma Z_i^2 - N \cdot \bar{Z}^2\end{aligned}$$

where we have used the fact that $\bar{Z} \equiv \Sigma Z_i / N$.

Substituting back in the expression for σ gives us

$$\sigma = \sqrt{\frac{\Sigma Z_i^2}{N} - \bar{Z}^2}$$

If you think about it and scratch out some flowcharts, you'll find that it is possible to define all the values of $G(I)$ in a single multinested loop using one other subscripted variable (taking on values of 1, 3, 2, 2, 2, 1, 2, 2, 2, 3, 5) for the limits of an inner loop.

Problems

1. Using a worst-case analysis, determine the nominal, maximum, and minimum values for the following functions:

   (a) $f = \dfrac{1}{2\pi} \sqrt{\dfrac{1}{LC}}$ (ac circuit resonance frequency)

   $L = .002 \pm .005$, $C = 1.5 \times 10^{-6} \pm .3 \times 10^{-6}$

   Answers: f (nominal) = 2905.76, f (maximum) = 3751.32, f (minimum) = 2372.54

   (b) $R = \dfrac{2 \cdot V_0^2 \cdot \sin\beta \cos\beta}{g}$ (range of a projectile, no air drag)

   $V_0 = 100 \pm 3$, $\beta = 45 \pm 1°$, $g = 32.2 \pm .1$

   Answers: R (nominal) = 310.559, R (maximum) = 330.498, R (minimum) = 291.123

(c) $\frac{1}{f} = (n-1)\left(\frac{1}{R_1} + \frac{1}{R_2}\right)$ (lens formula, focal length)

$n = 1.56 \pm .2$, $R_1 = 3.76 \pm .03$, $R_2 = 2.48 \pm .02$

Answers: f (nominal) = 2.6685, f (maximum) = 2.78955, f (minimum) = 2.55579

(d) $N = N_0 \cdot \exp[-.693(T/T_{1/2})]$ (nuclear decay rate)

$T_{1/2} = 1.65 \times 10^{10} \pm .01 \times 10^{10}$, $T = 3.8 \times 10^{11} \pm .4 \times 10^{11}$, $N_0 = 8.643 \pm .005$

Answers: N (nominal) = 1.01234E−06, N (maximum) = 5.92312, N (minimum) = 1.69335E−07

(e) $Z = \left[R^2 + \left(2\pi fL - \frac{1}{2\pi fC}\right)^2\right]^{1/2}$ (ac series circuit impedance)

$f = 60 \pm .1$, $R = 16.2 \pm 1.6$, $L = .30 \pm .05$, $C = 10 \times 10^{-6} \pm 3 \times 10^{-4}$

Answers: Z (nominal) = 153.021, Z (maximum) = 286.037, Z (minimum) = 73.0132

2. Using the Gaussian distribution [through $G(I)$] and the random number generator, derive some data from a polynomial. Use this data for a linear least squares fit using your program from Sec. C. Investigate the similarity of the fitted polynomial to the generating polynomial for various types of polynomials.

3. Use the Gaussian distribution and the random number generator to do a statistical distribution study for the functions in Prob. 1. Make sure that the "random" number generator starts in a different point in its sequence on subsequent runs of the program (RANDOMIZE is the usual command statement in BASIC, for example). No answers for this one; they vary with each run! Determine the average, maximum, and minimum values for the variables, as well as the standard deviations. Use N equal to 100 to 500 or 100 to 1000.

REFERENCES

BENNETT, WILLIAM RALPH, JR., *Scientific and Engineering Problem-solving with the Computer.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1976. Written by a co-inventor of the first gas laser, this large book is magnificent proof that computers are a uniquely powerful and interesting tool for learning about the physical world. Here the programmer can study languages, cryptology, space probes, the spread of disease, wave motion, detection of tape doctoring, electronics, and much more. BASIC is the language of the book.

BEVINGTON, PHILIP R., *Data Reduction and Error Analysis for the Physical Sciences.* New York: McGraw-Hill Book Company, 1969. Bevington gives particular emphasis to types of data reduction that are important in nuclear physics. He discusses the properties of various distribution functions (binomial, Poisson, Gaussian, and Lorentzian), least square fits to linear, nonlinear, and orthogonal functions with and without data weighting, and some data manipulation techniques such as smoothing. Various tests for goodness of fit (correlation coefficients) and the importance of additional terms are covered. He provides well-documented FORTRAN programs for most of the topics discussed.

DANIEL, C., and FRED S. WOOD, *Fitting Equations to Data.* New York: John Wiley & Sons, Inc., 1971. This is a study in depth of the techniques and problems involved in fitting linear and nonlinear functions to data. Two large FORTRAN programs are discussed; they are available on tape from outside sources.

EHRLICH, ROBERT, *Physics and Computers.* Boston: Houghton Mifflin Company, 1973. This text is intended for physics majors at the intermediate undergraduate level and particularly in the laboratory. Ehrlich treats electrostatics, electric circuits, wave equations, and Monte Carlo calculations using the FORTRAN language only. He also discusses the use of an experimental χ^2 to determine the probability that the proper function was chosen in a curve-fitting program.

GROSSBERG, ALAN B., *FORTRAN for Engineering Physics*. New York: McGraw-Hill Book Company, 1973. This two-volume work presents the use of a computer for data analysis in the general physics laboratory.

HOVANESSIAN, SHAHEN A., and LOUIS A. PIPES, *Digital Computer Methods in Engineering*. New York: McGraw-Hill Book Company, 1969. This introductory text covers eigenvalue problems, root-finding, interpolation, time-frequency analysis and the fast Fourier transform, and various techniques of differential equation solution. Programs in both FORTRAN and BASIC are presented.

LAFARA, ROBERT L., *Computer Methods for Science and Engineering*. Rochelle Park, New Jersey: Hayden Book Company, Inc., 1973. Although it is not as readable as it could be, this is a good introduction to all the standard data reduction techniques by a working computer scientist. LaFara treats sources of error and error studies, interpolation, roots of equations, simultaneous equation solution including a detailed discussion of the Gauss-Jordan method, curve-fitting, smoothing of data, and numerical differentiation and integration.

PECKHAM, HERBERT D., *Computers, BASIC, and Physics*. Reading, Mass.: Addison-Wesley Publishing Company, 1971. Written by a physicist, this is intended for the undergraduate physics curriculum. The elements of BASIC, as run on a Hewlett Packard minicomputer, are presented in the first 25 pages. Then Peckham discusses in some detail difference (derivative) formulas, numerical integration techniques, solution of ordinary and partial differential equations, matrices, and random number methods. He also includes a section on the computer in the laboratory. His flowchart symbols were the basis for the ones I have used.

SHAMPINE, L. F., and M. K. GORDON, *Computer Solution of Ordinary Differential Equations*. San Francisco: W. H. Freeman and Company, 1975. This is an in-depth study of the theory and practice of using the Adams methods for solving differential equations. The programs are given in the book and are also available from outside sources; the documentation is very good. The Adams methods approximate the solution of a differential equation by replacing a function of two variables (in a first-order differential equation) by a polynomial that has the same derivatives and then integrating the polynomial. As implemented here, the methods use a variable interval size and automatic determination of the proper size to produce a requested accuracy. These methods are preferred to the Runge-Kutta methods when the function is very complex and when numerous intermediate points are required.

CHAPTER IV PROBLEMS

A. General

B. Architecture-Mechanical Engineering

C. Aeronautical

D. Electronics

E. Physics

IV

Problems for Computer Solution

A. General

A-1. The Quadratic Equation

The standard form for a quadratic equation in the variable X is

$$AX^2 + BX^2 + C = 0$$

Solutions to this equation fall into one of three classes:

1. $B^2 - 4AC < 0$ (*no real roots*)
2. $B^2 - 4AC = 0$ (*two identical roots*): $X_1 = X_2 = -B/(2A)$
3. $B^2 - 4AC > 0$ (*two real roots*): $X_1 = \dfrac{-B - (B^2 - 4AC)^{1/2}}{(2A)}$

$$X_2 = \frac{-B + (B^2 - 4AC)^{1/2}}{(2A)}$$

In science the appearance of a quadratic equation usually signals the existence of two possible answers to a physical question. For example, if the air drag is negligible, a ball thrown straight up with a speed V_0 is at a height H after a time T has elapsed:

$$H = V_0T - \tfrac{1}{2}GT^2$$

where G is the acceleration due to gravity. But this equation can be written in the form

$$(\tfrac{1}{2}G)T^2 + (-V_0)T + H = 0$$

so that it is a quadratic equation in T and there are two possible values of time T for one value of height H. Physically, a class 1 solution indicates that the ball never gets that high; a class 2 solution indicates that the height is the highest point in the motion so that it occurs only once, and a class 3 solution indicates a height that occurs at two times, once on the way up and once on the way down.

Write a computer program to solve the general quadratic equation. (You may want to flowchart it first.) For a check, here are some representative numbers:

1. $G = 32.2$, $V_0 = 44.2$, $H = 18.0$ — *Answers*: $T_1 = .497335$, $T_2 = 2.24801$
2. $G = 32.0$, $V_0 = 16.0$, $H = 4.0$ — *Answer*: $T = 0.5$
3. $G = 32.16$, $V_0 = 7.8$, $H = 1.0$ — *Answer*: None

A-2. The Cubic Equation

If a quadratic equation suggests two possible answers to a physical question, then a cubic equation appears when there are *three* possible answers. Suppose we add an upward force that builds linearly with time (Force = $K \cdot T$) to the tossed ball of the previous problem. Then the ball could be at a particular height at three different times: on the way up, on the way down, and on the way back up after the force becomes stronger than the gravitational force. The equation of height as a function of time for this motion is

$$H = V_0 T - \tfrac{1}{2} G T^2 + \frac{K T^3}{6}$$

To solve this equation we look now into the general solution of a cubic equation; the standard form is

$$X^3 + AX^2 + BX + C = 0 \tag{IV-1}$$

and if we make the substitution

$$Y = X + \frac{A}{3} \tag{IV-2}$$

it can be reduced to the form

$$Y^3 + FY + G = 0 \tag{IV-3}$$

where the new coefficients F and G are given by

$$F = \frac{3B - A^2}{3} \tag{IV-4}$$

$$G = \frac{2A^3 - 9AB + 27C}{27} \tag{IV-5}$$

Then we define D in terms of F and G:

$$D^2 = \left(\frac{F}{3}\right)^3 + \left(\frac{G}{2}\right)^2 \tag{IV-6}$$

Now the solutions fall into one of three classes, much as for the quadratic equation.

$$1.\ D^2 = 0 \quad (\textit{two real roots}):\ Y_1 = 2\left(-\frac{G}{2}\right)^{1/3} \tag{IV-7}$$

$$Y_2 = -\left(-\frac{G}{2}\right)^{1/3} \tag{IV-8}$$

$$2.\ D^2 > 0 \quad (\textit{one real root}):\ Y_1 = -\left(\frac{G}{2} + D\right)^{1/3} - \left(\frac{G}{2} - D\right)^{1/3} \tag{IV-9}$$

$$3.\ D^2 < 0 \quad (\textit{three real roots}):\ Y_1 = 2\left(-\frac{F}{3}\right)^{1/2} \cos\frac{\phi}{3} \tag{IV-10}$$

$$Y_2 = 2\left(-\frac{F}{3}\right)^{1/2} \cos\frac{\phi}{3} + \frac{2\pi}{3} \tag{IV-11}$$

$$Y_3 = 2\left(-\frac{F}{3}\right)^{1/2} \cos\frac{\phi}{3} + \frac{4\pi}{3} \tag{IV-12}$$

$$\text{where } \cos\phi = \left(\frac{3G}{2F}\right)\left(-\frac{3}{F}\right)^{1/2} \tag{IV-13}$$

Note that in each case the solutions given are for Eq. IV-3; you must use Eq. IV-2 to obtain the roots to Eq. IV-1.

This may appear to be a formidable array of equations, but they are really quite easy to solve with a formula-oriented language such as BASIC or FORTRAN. There are two traps, however, that you may fall into if you are left to your own devices at this point. One trap appears when you try to obtain the cube root of a negative number with the exponentiation operator. *We* know that the cube root of a negative number is simply the negative of the cube root of the absolute value; for example $(-8)^{1/3} = -2 = -(|-8|)^{1/3}$. But in general the *computer* does not know that and will try to do the operation through logarithms—and quit and complain to you. A neat way to handle this problem is to use the SGN function (if available), which returns a +1 for a positive argument, a −1 for a negative argument, and 0 for a zero argument.

The second pitfall involves the finding of the angle ϕ after we have learned the value of $\cos\phi$ from Eq. IV-13. Lacking a built-in arc cosine function, you have to use the arc tangent function. You can do this by calculating $\sin\phi$ from $\cos\phi$ [$\sin\phi = (1 - \cos^2\phi)^{1/2}$] and then by using arc tan ($\sin\phi/\cos\phi$). But the arc cos operation yields an angle between 0 and π radians, whereas the arc tan operation gives an angle between $-\pi/2$ and $+\pi/2$. Therefore check the angle ϕ after obtaining it from arc tan ϕ, and add π to it if it is less than zero, so that ϕ ends up in the correct quadrant.

Here are some sample equations with intermediate values to help you debug your program.

$1.\ X^3 + 6X^2 + 9X + 4 = 0$

Intermediate values: $F = -3, G = 2, D^2 = 0, C = -1,$
$Y_1 = -2, Y_2 = 1$ *Answers*: $X_1 = -4, X_2 = -1$

2. $X^3 + 9.7X^2 + 3X - 29.7 = 0$
   Intermediate values: $F = -28.3633, G = 28.2054,$
   $D^2 = -646.214, \cos\phi = -.485119,$
   $\phi = -1.0643 \rightarrow 2.07729$
   $Y_1 = 4.73333, Y_2 = -5.76667,$
   $Y_3 = 1.03333$ *Answers*: $X_1 = 1.5, X_2 = -9, X_3 = -2.2$

3. $X^3 + \frac{14}{3}X^2 + \frac{13}{3}X - 2 = 0$
   Intermediate values: $F = -2.92593, G = -1.21262$
   $D^2 = -.560128, \cos\phi = .629479$
   $Y_1 = 1.88889, Y_2 = -1.44444,$
   $Y_3 = -.444444$ *Answers*: $X_1 = .333333, X_2 = -3, X_3 = -2$

4. $X^3 + 4.6X^2 + 6.85X - 21.35 = 0$
   Intermediate values: $F = -.203334, G = -24.6433$
   $D^2 = 151.822, Y_1 = 2.93319$ *Answer*: $X_1 = 1.39985$
   (*Note*: The correct answer is $X = 1.4$; the relatively large error occurs because d is almost the same in value as $G/2$, and thus their subtraction results in the loss of about one significant figure.)

Now you are ready to solve the tossed ball problem. If a height that the ball never reaches is chosen, then any real roots are negative. If the ball is thrown upward, then there can be three values for time; if you find a height for which there are only two different values, on the other hand, then you have found the maximum height for the ball just before it starts to fall back down, or the minimum height on its fall. But if the ball is thrown down (V_0 negative), it eventually comes back up, because the equation doesn't include the possibility of an obstacle in the ball's path. Consider the acceleration of gravity, G, to be equal to 32.2 ft/sec^2; the times are in seconds, the speeds are in ft/sec, and the units of K are ft/sec^3. The answers for height are then in feet.

1. $V_0 = 46.7, K = 1.46, H = 5.37$
   Answers: $T_1 = .119915, T_2 = 2.9146, T_3 = 63.1299$
2. $V_0 = 46.7, K = 1.47$. Maximum initial height = ?
   Answer: Maximum initial height is 34.6523 at $T = 1.5$
3. $V_0 = -46.7, K = 1.47, H = -50$
   Answers: $T_1 = 68.4552, T_2 = .833961$
4. $V_0 = -46.7, K = 489$. Lowest height = ?
   Answer: Lowest height is -17.1941 occurring at $T = .508$

A-3. Simultaneous Equations in Two Unknowns

Section B of Chap. III shows how to devise a program that solves any number of simultaneous equations, but it is worthwhile to write a separate program for the simple special case of two equations in two unknowns.

If the unknowns are X and Y, then the general form for the equations is

$$A_{11}X + A_{12}Y = A_{13}$$
$$A_{21}X + A_{22}Y = A_{23}$$

and the general solution is

$$X = \frac{A_{13}A_{22} - A_{23}A_{12}}{D}$$

$$X = \frac{A_{23}A_{11} - A_{13}A_{21}}{D}$$

where $D = A_{11}A_{22} - A_{12}A_{21}$

These equations appear in the mathematical description of nature when an event depends simultaneously on two separate processes. For example, the time and place where the sleek sports car of Fig. IV-1 passes the humble sedan ahead of it depends on their speeds and their positions. In fact, pass or crash will occur at a distance d and a time t given by

$$d = 4.32 + 69.9t$$

and

$$d = 8.10 + 55.5t$$

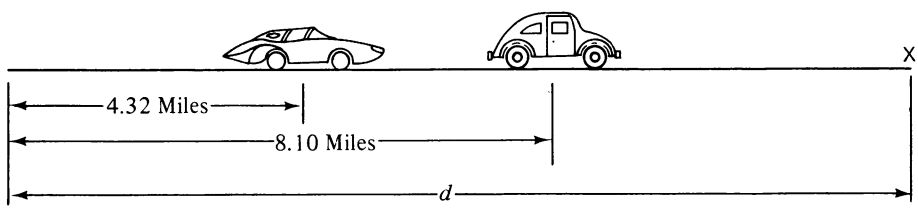


Figure IV-1.

Try these numbers for a check on your program; you should get $d = 22.1728$ miles and $t = .255405$ hours.

As another example, the current in loop 1 in Fig. IV-2 depends on the current in loop 2, because they share a common segment. With the help of Kirchhoff's laws, we can write

$$-V_1 + i_1R_1 + (i_1 + i_2)R_3 + V_2 = 0$$
$$-V_3 + i_2R_2 + (i_1 + i_2)R_3 + V_2 = 0$$

which can be put in the standard form:

$$i_1(R_1 + R_3) + i_2R_3 = V_1 - V_2$$
$$i_1R_3 + i_2(R_2 + R_3) = V_3 - V_2$$

Try these circuit values: $R_1 = 40$ ohms, $R_2 = 61$ ohms, $R_3 = 72$ ohms, $V_1 = 41$ volts, $V_2 = -23$ volts, and $V_3 = 32$ volts. You should get $i_1 = .468698$ amperes and $i_2 = .159802$ amperes.

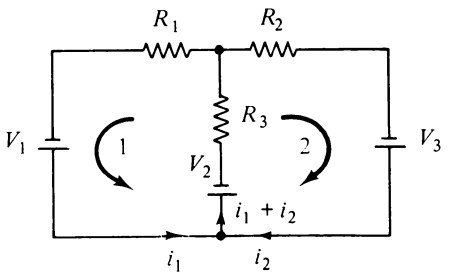


Figure IV-2.

A-4. Plotting of Data on the Printer

A high-quality X-Y plotter is best, but lacking that you can still produce some very useful graphic output on the printer. There is no equal for the ability of a graph to quickly show you how well you have fitted a curve to your data, for example.

Most graphs require that the data be *scaled* to fit it in the available space, so this is the first task. Thus, if you have 61 available print positions (which means 60 intervals) and the numbers to be plotted vary from 0 to 120, then your scale factor is one-half. Also, the origin must be shifted if the data doesn't have a minimum of (0,0).

If you have N available print positions for plotting data listed under the variable Y, and if you choose

$Y1$ = value of Y at the bottom left-hand corner of the graph (the minimum plot value of Y)

$Y2$ = value of Y at the upper left-hand corner of the graph (the maximum plot value of Y)

then the scaling and shifting requirements are met by making the following calculations:

$$K \equiv \text{Scale factor} = \frac{N - 1}{Y2 - Y1} \tag{IV-14}$$

$$M \equiv \text{Print position} = \text{Integer value of } [K(Y - Y1) + .5)] \tag{IV-15}$$

Suppose that we have 61 print positions and that we have Y values that vary from -6 to $+15$; we might then choose $Y1$ equal to -10 and $Y2$ equal to $+20$. This would give us a scale factor of

$$K = \frac{61 - 1}{20 - (-10)} = 2$$

and the print positions would vary between

$$M = \text{Integer value of } \{2[-6 - (-10)] + .5\} = 8$$

and

$$M = \text{Integer value of } \{2[20 - (-10)] + .5\} = 60$$

where the minimum and maximum values for Y have been substituted into Eq. IV-15. (The .5 that is added ensures that M is the integer *closest* to the calculated number.)

The next step is to send the printer to the print position specified by M for each data point and have it print a character such as the asterisk. In BASIC the statement

```
PRINT TAB(M-1); "*"
```

accomplishes this. In FORTRAN you can define a subscripted variable that can take on as many values as there are print positions; then for each point, make all these values a blank, but make the one with an index value given by M equal to the asterisk; finally, print out the variable under the A format. (Note that you can start the plotting to the right of the left-hand margin by adding the desired number of spaces to the number calculated from Eq. IV-15.)

In manual plotting, the usual procedure is to choose $Y1$ and $Y2$ so that values intermediate to the plotted values are easily estimated from the graph; this requires that the difference between $Y1$ and $Y2$ be related simply to the number of intermediate divisions. You should do the same thing in computer plotting by thoughtful choice of $Y1$ and $Y2$. Thus, in the example, $Y1$ was chosen as -10, in recognition of our decimally based number system, and $Y2$ was chosen to make the difference equal to one-half the number of intervals (which made the scale factor a single digit number).

For data plotting, the X values (in the direction of the line feed) should have uniform spacing. (You could always compensate for irregular spacing by making the interval very small, but this might result in a graph many feet long!)

It is suggested that you try your program with just a few points to get it debugged and then go on to print labels and values along the axis, a grid network inside the plotting area, etc. You can also try this slightly larger data set: $X1 = 7.4$, $Y1 = 914$; $X2 = 7.9$, $Y2 = 836$; $X3 = 8.4$, $Y3 = 407$; $X4 = 8.9$, $Y4 = 428$; $X5 = 9.4$, $Y5 = 516$; $X6 = 9.9$, $Y6 = 536$; $X7 = 10.4$, $Y7 = 557$; $X8 = 10.9$, $Y8 = 602$; $X9 = 11.4$, $Y9 = 710$; and $X10 = 11.9$, $Y10 = 850$.

A-5. Plotting Functions on the Printer

It should not be too difficult to modify your program of subsection A-4 so that you can plot functions. Define the function in a single statement so that it will be easy to switch functions.

If you wish to plot the function between a minimum value, $X = X1$, and a maximum value, $X = X2$, then you can start at $X1$ and successively increment X by

$$\Delta X = \frac{X2 - X1}{N1}$$

where $N1$ is the number of points you wish plotted.

You may wish to have the program determine the maximum and minimum values for the function before beginning the plotting; then it can also determine $Y1$, $Y2$, and K. To obtain the extrema, assume that the first functional value of Y is both the maximum and minimum and then replace each one as necessary as you step through the remaining values of X.

If you are plotting distances to scale or if you want to preserve the shape of a geometrical function, you need to compensate for the fact that the printer advances more between lines than between adjacent characters. For a Teletype this ratio is about 5 to 3,

but a line printer may have the ratio closer to 8 to 5; your scale factors should be in the inverse ratio.

Some suggested functions for plotting are a driven oscillator, $Y = X \cdot \sin(X)$ and a damped oscillator, $Y = 70e^{-.8X} \cos(10X)$.

A-6. Rounding Off Calculated and Printed Answers

Your computer may already know how to round off printed answers, but many do not. Even if yours does, plan to read on if you ever expect to buy a house.

If your input data are accurate to only two significant figures, then any quantity calculated from them will be similarly limited in accuracy, and there is no point in displaying meaningless numbers. Or, if you're doing calculations involving money, you'll want to avoid splitting pennies.

This rounding off can be accomplished by a calculation of the general form

$$A = \frac{\text{Integer value of } [(10^M \cdot A) + .5]}{10^M}$$

where the A inside the parentheses is the number we wish to round off, and M is a positive or negative integer. The .5 does the rounding; without it there would be simple truncation.

When M is a positive integer, this statement rounds off A so that it has only M digits to the right of the decimal point. Let's try it for A equal to 0.624737 and M equal to 3:

$$A = \frac{\text{Integer value of } [(10^3 \cdot .624737) + .5]}{10^3}$$

$$= \frac{\text{Integer value of } (624.737 + .5)}{10^3}$$

$$= \frac{625}{1000} = .625$$

So it does perform as advertised. Now look what happens when M is negative: Try A equal to 7047.4 and M equal to -1. Then

$$A = \frac{\text{Integer value of } [(10^{-1} \cdot 7047.9) + .5]}{10^{-1}}$$

$$= \frac{\text{Integer value of } (704.74 + .5)}{10^{-1}} = \frac{705}{10^{-1}} = 7050$$

and you can see that negative M rounds off a number to the left of the decimal point. If we had tried M equal to -4 with this number, the equation would have converted A to 10000; if M were any more negative, though, the converted value would be zero. With a slight modification of the general form you can use this last property to determine the relative size of a number so that you can print out any desired number of digits.

A good warm-up problem is to calculate the Celsius equivalent of Fahrenheit temperatures from 32°F to 212°F in 20° increments. The formula is

$$°C = \tfrac{5}{9}(°F - 32°)$$

and each of the printed answers should have just three digits.

Then suppose you have borrowed $25,000 to buy an old house and have agreed to pay 8.75% yearly interest on the principal. Each month you pay the bank an amount such that the loan is repaid in 20 years. Each time a payment is made, the excess of monthly payment over interest is subtracted from the principal.

Each time, the value of the interest must be rounded off to the nearest cent. Thus the first interest payment is the rounded value of

$$I_1 = \frac{.0875}{12} (25000)$$

and, for a monthly payment of N dollars, the new principal is

$$P_1 = 25000 - (N - I_{1,\text{rounded}})$$

Similarly, the second payment results in a new interest charge of

$$I_2 = \frac{.0875}{12} \cdot P_1$$

and the new principal amount is

$$P_2 = P_1 - (N - I_{2,\text{rounded}})$$

Naturally you want to find out if you can afford this house, and for that you'll need to know the size of the monthly payment. Assume that the loan is for 20 years (240 payments) and that all the monthly payments are the same except for the last one, which can be smaller but only by the least possible amount. Print out the monthly payment amount, the total interest paid, and the amount of the last payment. *Hint*: The monthly payment is between $200 and $250, and the last payment is $219.11.

A-7 Ordering Numbers by Magnitude

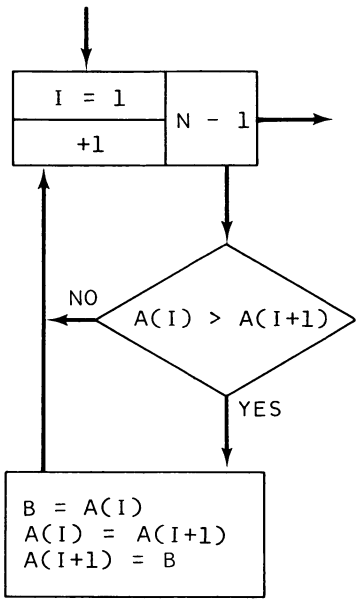
Suppose that you have a set of data [$A(I)$, I = 1 to N] that you wish to rearrange so that the numbers are in order of increasing magnitude. Thus for the simple group of 4 numbers

$$A(1) = 5$$
$$A(2) = 3$$
$$A(3) = 6$$
$$A(4) = 1$$

you wish to change the array to

$$A(1) = 1$$
$$A(2) = 3$$
$$A(3) = 5$$
$$A(4) = 6$$

To get the computer to do this, you should ask yourself how you would teach a simpleton to do the job. First you might teach him how to get the largest number to the bottom of the list. He could do this by comparing the first two terms and switching them if the first is larger than the second, then by comparing the second and third and switching them if the second is larger, etc. The flowchart of Fig. IV-3 shows how this can be done. Note how B is used to save the old value of $A(I)$ so it isn't lost in the shuffle.



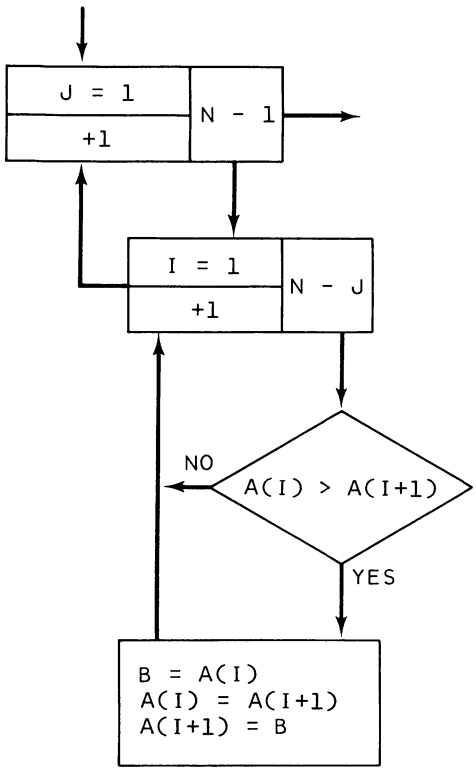
Example solution

	$I = 1$	$I = 2$	$I = 3$
$A(1) = 5$	→ 3	→ 3	→ 3
$A(2) = 3$	→ 5	→ 5	→ 5
$A(3) = 6$	→ 6	→ 6	→ 1
$A(4) = 1$	→ 1	→ 1	→ 6

Figure IV-3.

The next step for our dimwit is to go through the list again to get the second largest number to the next-to-the-last position, and so on. Fig. IV-4 shows how all this can be accomplished, including the first step.

Now you should be ready to write out a program, debug it with the numbers of the example, and try it with at least one other set of numbers. With a simple change, your program will arrange data in order of *decreasing* magnitude, so do it. If you are programming in Extended BASIC, you can probably use the same logic and string variables to alphabetize a list of names.



Example solution

$J = 1$ $I = 1$ $I = 2$ $I = 3$

$A(1) = 5 \rightarrow 3 \rightarrow 3 \rightarrow 3$
$A(2) = 3 \rightarrow 5 \rightarrow 5 \rightarrow 5$
$A(3) = 6 \rightarrow 6 \rightarrow 6 \rightarrow 1$
$A(4) = 1 \rightarrow 1 \rightarrow 1 \rightarrow 6$

$J = 2$ $I = 1$ $I = 2$

$A(1) = 3 \rightarrow 3$
$A(2) = 5 \rightarrow 1$
$A(3) = 1 \rightarrow 5$
$A(4) = 6 \rightarrow 6$

$J = 3$ $I = 1$

$A(1) = 1$
$A(2) = 3$
$A(3) = 5$
$A(4) = 6$

Figure IV-4.

A-8. Perpetual Calendar and Time Periods

In legal and business matters (particularly loans), it is often necessary to determine when a prescribed time period will be up. To determine this terminal date, it is necessary to know the beginning date and the time period and to note that (a) January, March, May, July, August, October, and December all have 31 days, (b) April, June, September, and November have 30 days, and (c) February has 29 days when (i) the year is an integer multiple of 4, but not of 100, or (ii) the year is an integer multiple of 400. [Thus 1972 was a leap year by criterion (i), and the year 2000 will be a leap year by criterion (ii).] Otherwise, February has 28 days.

Assume that the input data will be in the form

6, 30, 1972, 1, 0, 10

which would be interpreted as meaning that the beginning date is June 30, 1972, and that the time period involved is 1 year, 0 months, and 10 days.

Your output should be in the form

```
FOR A BEGINNING DATE OF JUNE 30, 1972, AND A TIME PERIOD OF 1 YEAR,
0 MONTHS, AND 10 DAYS, THE TERMINAL DATE IS JULY 10, 1973.
```

A flowchart as your first step comes highly recommended.

A-9. Vending Machine Logic

If you've ever directed derogatory remarks (or worse) toward vending machines, then you may gain more sympathy for them after seeing what they go through.

Suppose that each of your customers is represented by a string of numbers. The initial numbers are to have the following significance:

1 — A penny has been inserted in your coin slot and must be returned.
5 — A nickel has been inserted in your coin slot.
10 — A dime has been inserted in your coin slot.
13 — A slug has been inserted in your coin slot.
25 — A quarter has been inserted in your coin slot.

A number 9 after one or more of these numbers means that your customer is now pulling on one of your knobs to try to obtain the particular pack of gum or candy bar that he wishes. The next number indicates the price (in cents) of the item represented by that knob. Your task is to give him candy and any necessary change if he has played the game according to the rules, but only return his coins with an appropriate comment if he has given you a penny or a slug. Thus for a data string of 25, 9, 20, you wish to print out something like

HERE IS YOUR CANDY AND 05 CENTS CHANGE

whereas for a data string of 5, 10, 25, 9, 50, you would respond in a fashion such as

NO! TOO LITTLE MONEY

Hint: In devising a program to do this, you will find that a flowchart really sweetens the way. Be sure to test your flowchart and then your program for all the indicated possibilities.

A-10. Least Squares Fit of Data to a Straight Line

No student of the physical sciences who has taken introductory laboratories has ever escaped the imperious command, "Plot your experimental data on a graph, draw the best straight line through the points, and determine . . . from the slope of the line." The drawn line is to represent a visual *average* of numerous measurements and thereby should be more accurate and more meaningful than simply using two data points. Theory will have predicted that the relationship between dependent and independent variables is a linear one; but, even if it is, various types of experimental errors will conspire to prevent the data from falling exactly on a straight line. You must then use your own judgment (or fertile imagination) to draw a straight line such that equal numbers of points fall on each side of the line (or such that the desired answer is obtained). However, your opportunities for creativity in draftsmanship can be effectively stifled by replacing you with a machine (the computer). The fitting problem is treated in great generality in Secs. C and D of Chap. III, but a separate program for this special and ubiquitous case is worthwhile and instructive.

The formulas for this analysis can be derived rather easily from the calculus, but you can use the results even if you can't follow the derivation. We assume that N measurements

of Y have been made $\{Y_{Exp}(I), I = 1 \text{ to } N\}$ by causing the independent variable X to take on N values $\{X(I), I = 1 \text{ to } N\}$. Then our goal is to fit these points to a straight line, the equation for which is

$$Y_{Calc}(I) = A1 + A2 \cdot X(I)$$

The constants $A1$ and $A2$ are the Y intercept and the slope of the line, respectively, and are to be calculated in a way that satisfies the criterion of least squares. This means that we wish to minimize the sum

$$S = \sum_{I=1}^{N} \{Y_{Calc}(I) - Y_{Exp}(I)\}^2$$

(Note that we are tacitly assuming that all the uncertainty is in the dependent variable; because this usually isn't the case, the deviations from the derived line may appear somewhat greater than expected.) The minimum for this sum must be where it, as a function of $A1$ and $A2$, has *zero* slope. The slopes are $\partial S/\partial A1$ and $\partial S/\partial A2$, so we obtain

$$\begin{aligned} \frac{\partial S}{\partial A1} &= \frac{\partial}{\partial A1} \Sigma\{A1 + [A2 \cdot X(I)] - Y_{Exp}(I)\}^2 \\ &= 2 \Sigma\{A1 + [A2 \cdot X(I)] - Y_{Exp}(I)\} \\ &= 0 \end{aligned}$$

or

$$(A1 \cdot N) + [A2 \cdot \Sigma X(I)] = \Sigma Y_{Exp}(I) \qquad \text{(IV-16)}$$

$$\begin{aligned} \frac{\partial S}{\partial A2} &= \frac{\partial}{\partial A2} \Sigma\{A1 + [A2 \cdot X(I)] - Y_{Exp}(I)\}^2 \\ &= 2 \Sigma\{A1 + [A2 \cdot X(I)] - Y_{Exp}(I)\} \cdot X(I) \\ &= 0 \end{aligned}$$

or

$$[A1 \cdot \Sigma X(I)] + [A2 \cdot \Sigma X(I)^2] = \Sigma X(I) \cdot Y_{Exp}(I)] \qquad \text{(IV-17)}$$

We can simplify these equations if we make the definitions

$$\overline{X} = \Sigma X(I) \qquad \text{(IV-18)}$$

$$\overline{Y} = \Sigma Y_{Exp}(I) \qquad \text{(IV-19)}$$

$$\overline{XY} = \Sigma X(I) \cdot Y_{Exp}(I)] \qquad \text{(IV-20)}$$

$$\overline{X2} = \Sigma X(I)^2 \qquad \text{(IV-21)}$$

Then Eqs. IV-16 and IV-17 can be written, respectively, as

$$(N \cdot A1) + (\bar{X} \cdot A2) = \bar{Y} \tag{IV-22}$$

and

$$(\bar{X} \cdot A1) + (\overline{X2} \cdot A2) = \overline{XY} \tag{IV-23}$$

so that finding the "best" straight line fit to the data has been reduced to finding the coefficients $A1$ and $A2$ from these two simultaneous equations. Solving them by substitution yields

$$A2 = \frac{(N \cdot \overline{XY}) - (\bar{X} \cdot \bar{Y})}{(N \cdot \overline{X2}) - \bar{X}^2} \tag{IV-24}$$

$$A1 = \frac{\bar{Y} - (A2 \cdot \bar{X})}{N} \tag{IV-25}$$

Determining the "goodness of fit" is best accomplished by having the computer plot both the data and the straight line. The second best approach is to print out $X(I)$, $Y_{\text{Exp}}(I)$, $Y_{\text{Calc}}(I)$, and $Y_{\text{Calc}}(I) - Y_{\text{Exp}}(I)$ so that you can be sure that the input data were read in properly and so that you can spot any points that are unreasonably far from the derived straight line. Finally, you should calculate the average deviation from the line, defined by

$$\delta = \sqrt{\frac{S}{N}} \tag{IV-26}$$

Now for an application. A simple laboratory method for determining the capacitance (C) of a particular capacitor is to charge it with a battery and then measure the voltage (V) as a function of time (T) as it discharges through a known resistance (R). The theoretical relationship is

$$V = V_0 \exp(-T/RC)$$

where V_0 is the measured voltage when timing is begun. Taking natural logarithms of both sides of this equation, we obtain the equation of a straight line:

$$\ln V = (-1/RC)T + \ln V_0$$

The slope of the line should then be $(-1/RC)$, and for known R, we can calculate C. But this actually assumes that T is measured in seconds; if T is measured in minutes, then the slope should be $(-60/RC)$.

Here is some unfudged laboratory data for your program:

Time in minutes(T)	*Voltage in volts (V)*	*Leakage resistance in ohms (R)*
0.0	1.50	4.8×10^5
1.0	1.36	
2.0	1.23	
3.0	1.15	
4.0	1.017	
5.0	.925	
6.0	.840	
7.0	.760	
8.0	.695	
9.0	.630	
10.0	.575	
11.0	.527	
12.0	.480	

Using this data you should get δ equal to 8.16953×10^{-3} and a value for C (rounded to two significant figures) of 1.3×10^{-3} farads. (The marked value on the laboratory capacitor was 1×10^{-3} farads.)

A-11. Least Squares Fit of Data to a Parabola

We can easily extend the least squares analysis of the previous problem to one more dimension if we are willing to relinquish some generality in fitting our data to a parabola that passes *through* the origin. This means that we must have confidence that the value of the dependent variable (Y) when the independent variable (X) is zero is either (a) zero, or (b) accurately known. In the second case, a new dependent variable must be defined. For example, if $Y(0) = 4.00$, then we could define

$$Z(X) = Y(X) - 4.00$$

and then the values of $Z(X)$ could be fitted to a parabola through the origin.

We assume again that N measurements of Y have been made $\{Y_{\text{Exp}}(I),\ I = 1 \text{ to } N\}$ by causing the independent variable X to take on N values $\{X(I), I = 1 \text{ to } N\}$. Then we wish to fit these points to the parabola defined by

$$Y_{\text{Calc}}(I) = [A1 \cdot X(I)] + [A2 \cdot X(I)^2]$$

Again we form the sum of the squares of the deviations

$$S = \sum_{I=1}^{N} \{Y_{\text{Calc}}(I) - Y_{\text{Exp}}(I)\}^2$$

and then minimize this sum with respect to the coefficients $A1$ and $A2$:

$$\frac{\partial S}{\partial A1} = \frac{\partial}{\partial A1}\Sigma\{[A1 \cdot X(I)] + [A2 \cdot X(I)^2] - Y_{\text{Exp}}(I)\}^2$$
$$= 2\Sigma\{[A1 \cdot X(I)] + [A2 \cdot X(I)^2] - Y_{\text{Exp}}(I)\} \cdot X(I)$$
$$= 0$$

or

$$[A1 \cdot \Sigma X(I)^2] + [A2 \cdot \Sigma X(I)^3] = \Sigma[Y_{\text{Exp}}(I) \cdot X(I)] \tag{IV-27}$$

and

$$\frac{\partial S}{\partial A2} = \frac{\partial}{\partial A2}\Sigma\{[A1 \cdot X(I)] + [A2 \cdot X(I)^2] - Y_{\text{Exp}}(I)\}^2$$
$$= 2\Sigma\{[A1 \cdot X(I)] + [A2 \cdot X(I)^2] - Y_{\text{Exp}}(I)\} \cdot X(I)^2$$
$$= 0$$

or

$$[A1 \cdot \Sigma X(I)^3] + [A2 \cdot \Sigma X(I)^4] = \Sigma[Y_{\text{Exp}}(I) \cdot X(I)^2] \tag{IV-28}$$

We can simplify the appearance of these equations by defining

$$\overline{X} = \Sigma X(I) \tag{IV-29}$$
$$\overline{Y} = \Sigma Y_{\text{Exp}}(I) \tag{IV-30}$$
$$\overline{XY} = \Sigma[X(I) \cdot Y_{\text{Exp}}(I)] \tag{IV-31}$$
$$\overline{X2Y} = \Sigma[X(I)^2 \cdot Y_{\text{Exp}}(I)] \tag{IV-32}$$
$$\overline{X2} = \Sigma X(I)^2 \tag{IV-33}$$
$$\overline{X3} = \Sigma X(I)^3 \tag{IV-34}$$
$$\overline{X4} = \Sigma X(I)^4 \tag{IV-35}$$

Then Eqs. IV-27 and IV-28 can be written, respectively, as

$$(\overline{X2} \cdot A1) + (\overline{X3} \cdot A2) = \overline{XY} \tag{IV-36}$$

and

$$(\overline{X3} \cdot A1) + (\overline{X4} \cdot A2) = \overline{X2Y} \tag{IV-37}$$

From these simultaneous equations the coefficients for our parabola can be determined by substitution; the results are

$$A1 = \frac{(\overline{X2Y} \cdot \overline{X3}) - (\overline{XY} \cdot \overline{X4})}{\overline{X3}^2 - (\overline{X2} \cdot \overline{X4})} \tag{IV-38}$$

$$A2 = \frac{(\overline{XY} \cdot \overline{X3}) - (\overline{X2Y} \cdot \overline{X2})}{\overline{X3}^2 - (\overline{X2} \cdot \overline{X4})} \tag{IV-39}$$

Here is some Cooke titration data for use in checking your program.

Acidity	*Carbon dioxide added*
8.815	0.000
8.810	0.076
8.500	0.151
8.010	0.226
7.615	0.300
7.253	0.375
6.980	0.449
6.740	0.523
6.605	0.596
6.490	0.669
6.400	0.742

Answers: $A1 = -4.29407, A2 = 1.14921, \delta = .16$

A-12. How Fast Will It Go?

Assuming that you completed Sec. D of Chap. III, the computer can help you determine the top speed of your motorized vehicle without tarnishing either your driving record or your nerves.

The resisting force on a vehicle, which determines the power required for a particular speed, is composed of at least two terms.[1] One term is the rolling and sliding resistance of tires and mechanical parts, and the other is the air drag. The first (kinetic friction) is approximately independent of speed, and the second is proportional to the square of the speed. If the vehicle's engine is disengaged after reaching some speed, then the deceleration is due to this total resisting force, and from Newton's Second Law we can write

$$\text{Drag force} = -C1 - (C2 \cdot V^2) = M \cdot A \tag{IV-40}$$

where M is the mass of the vehicle, A is the (negative) acceleration, and $C1$ and $C2$ are the drag constants we wish to determine. Tables of integrals can be used to integrate Eq. IV-40, and this yields the speed as a function of time:

$$V = K1 \cdot \tan\left\{\arctan\left(\frac{V0}{K1}\right) - (K2 \cdot T)\right\} \tag{IV-41}$$

Here $V0$ is the speed when timing is begun ($T = 0$), V is the speed at a later time, and the constants $K1$ and $K2$ are related to the earlier constants by

$$C1 = K1 \cdot K2 \cdot M \tag{IV-42}$$

[1] If you are interested in a more rigorous treatment of this subject, you will want to look at the article by Gene I. Rochlin, "Drag Force on Highway Vehicles," *Am. J. Phys.*, Vol. 44, No. 10 (October 1976), p. 1010, and the references given there.

and

$$C2 = \frac{K2 \cdot M}{K1} \tag{IV-43}$$

If we can determine $K1$ and $K2$ in Eq. IV-41, then we can estimate the required horsepower as a function of speed and the top speed of the tested vehicle. This follows from the fact that the power being used at a particular speed is the resisting force of Eq. IV-40 multiplied by the speed. Thus we can write

$$\text{Power required} = (C1 \cdot V) + (C2 \cdot V^3) \tag{IV-44}$$

and the top speed is that speed for which the power required is equal to the maximum power available (assuming that the vehicle is geared to deliver that power at the top speed):

$$(C1 \cdot V_{\text{Top}}) + (C2 \cdot V_{\text{Top}}^3) = \text{Power available} \tag{IV-45}$$

One thorn in this sequence is the question of units. The equations are fine as long as the units are consistently those of the British engineering system or the metric system. But here it is most convenient to use units of miles/hr rather than ft/sec for the speed and horsepower rather than ft-lb/sec for power. To use these units, you need the conversion factors

$$K = 550 \frac{\text{ft-lb/sec}}{\text{horsepower}} \tag{IV-46}$$

and

$$K0 = \frac{5280 \text{ ft/sec}}{3600 \text{ miles/hr}} \tag{IV-47}$$

If our speed is measured in miles/hr and the time is in seconds, then the least squares fitting procedure, using Eq. IV-41, gives us a $K1$ and a $K2$ that are related to the constants of Eq. IV-40 by

$$C1 = K0 \cdot K1 \cdot K2 \cdot M \tag{IV-48}$$

and

$$C2 = \frac{K2 \cdot M}{K0 \cdot K1} \tag{IV-49}$$

The power in horsepower required at any speed is given by

$$\text{Power required} = \frac{(C1 \cdot V) + (C2 \cdot V^3)}{K} \tag{IV-50}$$

where the speed must be in ft/sec; the conversion of the speed to ft/sec is given by

$$V(\text{ft/sec}) = K0 \cdot V \text{ (miles/hr)} \tag{IV-51}$$

The top speed in ft/sec is given by solving (by the method of subsection A-2 or by trial and error) the equation

$$(C1 \cdot V_{\text{Top}}^3) + (C1 \cdot V_{\text{Top}}) - (K \cdot \text{Available horsepower}) = 0 \qquad \text{(IV-52)}$$

Really accurate data requires flat ground and no wind; Ohio is loathe to provide either of them, but you can check your program with these numbers obtained by testing a 1976 Volkswagen Rabbit:

Maximum horsepower available: 71 hp

$$\textit{Mass} = \text{Test weight}/g = \frac{2400 \text{ lb}}{32 \text{ ft/sec}^2} = 75 \text{ slugs}$$

Speed (miles/hr)	*Time (seconds)*
60	0.0
55	5.0
50	11.4
45	17.9
40	25.6
35	34.4
30	45.2

For these data my fitting program yields a $K1$ equal to 21.1513 and a $K2$ equal to 5.84928×10^{-3} for a χ of .598756. The $C1$ and $C2$ derived from Eqs. IV-48 and IV-49 are 13.609185 and 0.01414149, respectively. The estimated power required at 60 miles/hr from Eqs. IV-50 and IV-51 is 19.7 horsepower, and the estimated top speed from Eq. IV-52 is 138 ft/sec or 94 miles/hr.

B. Architecture-Mechanical Engineering

B-1. Data Search, Linear Interpolation, and Solar Radiation

Formula-oriented computer languages make it so easy to obtain values from equations that it is easy to forget that much engineering information is still contained in the form of tables of numbers; the user is expected to interpolate between listed values to determine any intermediate values that might be needed. The general polynomial fitting program described in Sec. C of Chap. III could certainly be utilized to obtain a formula or formulas that fit the data, but they might require more effort or be more difficult to update than the method given here.

We wish to store a table of data in the computer and then write a program that will access this data, using interpolation for intermediate values if necessary. We can begin by choosing the data table and by then asking ourselves very specifically how we would manually use the table. Suppose that for heating and cooling calculations we need to be able to find the normal radiation from the sun as a function of its angle above the horizon. Here is a table of such data:

Angle, degrees:	0	5	10	15	20	25	30	35	40	45	50
Radiation, Btu/hr/ft^2:	0	67	123	166	197	218	235	248	258	266	273
Angle:	60	70	80	90							
Radiation:	283	289	292	294							

If we were asked to determine the normal radiation for a particular angle from this table, we would first check to see that the angle was meaningful (i.e., between 0° and 90°). Next we would check to see if the angle happened to be one in the table; if so, the answer should be read directly; if not, some interpolation formula must be used. To derive a linear interpolation formula, suppose we look at the procedure for the specific value of 47°.

Our first step is to find angles that bracket the desired angle; in this case they are 45° and 50° with radiation values of 266 Btu/hr/ft^2 and 273 Btu/hr/ft^2 respectively. Then we set up a ratio based on the interval size for the angle and the radiation:

$$\frac{\text{Radiation at } 47^\circ - \text{Radiation at } 45^\circ}{47 - 45} = \frac{273 - 266}{50 - 45}$$

or

$$\text{Radiation at } 47^\circ = \frac{47 - 45}{50 - 45} \cdot (273 - 266)] + \text{Radiation at } 45^\circ$$
$$\simeq 269 \text{ Btu/hr/ft}^2$$

So now your task is to devise an algorithm and program that accomplishes these steps in general. A flowchart, checked with the example given, is a fine way to start. The data should be stored under a subscripted variable name.

B-2. Seating Capacity of a Room

The number of chairs or seats that can be installed in a room depends on geometric factors such as the size of the room, size of the chairs, aisle width, etc., and lends itself to computer calculation. Fig. IV-5 shows the important geometric dimensions.

The number of seats along a row will be the largest integer no larger than N, where

$$N \equiv \frac{W - (A1 + A2)}{C}$$

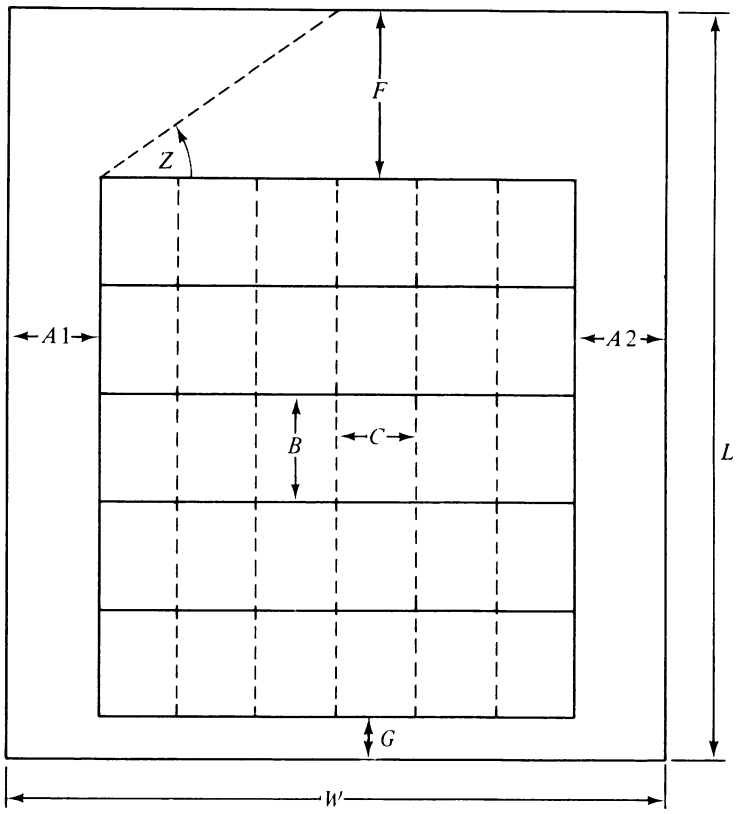
If N is not an integer, there will be some extra space available, and this should be added to the aisle widths: Add equally to the widths, except add it all to the other aisle if one of the widths is zero.

Next you should determine if the specified minimum viewing angle Z requires a greater front clearance than that specified initially by F; increase F if this is the case. The front clearance distance implied by the angle Z is $F2$, where

$$F2 \equiv \left[\frac{W - (A1 + A2)}{2}\right] \cdot \tan(Z)$$

and the aisle widths are the adjusted values.

Now the number of rows of seats can be determined in a fashion similar to the determination of number of seats in a row. Add any excess to the front clearance distance.



*A*1 = left aisle width (min)
*A*2 = right aisle width (min)
B = distance between adjacent rows
C = distance between adjacent seats
F = front clearance distance (min)
G = back clearance distance
L = length of room
W = width of room
Z = viewing angle from front end seat (min)

Figure IV-5. Seating capacity of a room

Here are the sample data and answers (distances are in feet).

Data:	*A*1 (min)	*A*2 (min)	*B*	*C*	*F* (min)	*G*	*L*	*W*	*Z* (degrees)
1.	3.5	3.5	3.2	1.8	11	6	43	23.7	35
2.	0	2.1	4.0	2.0	3.5	0	24	21.0	30

Answers:	*A*1	*A*2	*F*	No. seats in row	No. of rows	Total no. of seats
1.	3.75	3.75	13	9	8	72
2.	0	3.0	8.0	9	4	36

B-3. Wind-Induced Stress on Buildings

From Bernoulli's fluid energy relation, the pressure P on the side of a building due to a wind with speed V blowing against it is given by

$$P = \frac{\rho V^2}{2}$$

where ρ is the density of the air. Once we know the pressure at a certain point on the side of a building, then the force there is the area times the pressure. The difficulty is that the wind speed increases with altitude above the ground; an empirical formula for this variation is[2]

$$V = V0 \cdot \left(\frac{H}{H0}\right)^{1/7}$$

where V is the wind speed at a height H if $V0$ is a known windspeed at a lower height $H0$. If now we assume that the wind speed is constant from the ground level to the height $H0$ and from there varies according to this formula, then the summation of pressures times small areas can be accomplished with the calculus, yielding

$$F = \tfrac{7}{9} \cdot Q0 \cdot W \cdot H0 \cdot \left[\left(\frac{H1}{H0}\right)^{9/7} + \tfrac{2}{7}\right]$$

$$M = \tfrac{7}{16} \cdot Q0 \cdot W \cdot H0^2 \cdot \left[\left(\frac{H1}{H0}\right)^{16/7} + \tfrac{1}{7}\right]$$

where

$$Q0 = \frac{\rho V0^2}{2}$$

In these equations F is the total force on the side of a building of height $H1$ and width W; M is the total overturning moment due to that force; $Q0$ is the dynamic pressure due to air of density ρ and speed $V0$ at height $H0$. If we wish to measure the speed in miles per hour and the heights in feet, then we can obtain the force in pounds and the overturning moment in foot-pounds by calculating $Q0$ from

$$Q0 = (2.5565 \times 10^{-3}) \cdot V0^2$$

where standard sea-level air density has been assumed.

It is revealing to see how large these forces and moments can be. A certain university two-story science building has a width of 120 feet and a height of 40 feet; a library tower has a width of 125 feet and a height of 150 feet. Assume a 90-mile-per-hour gale wind and an $H0$ equal to 10 feet, and determine the force and overturning moment on a side of each of these buildings.

Answers: Forces are 120,402 pounds and 660,409 pounds, respectively; the moments are 2.60033×10^6 foot-pounds and 5.52529×10^7 foot-pounds, respectively.

[2] Henry J. Cowan, *Architectural Structures* (New York: American Elsevier Publishing Company, Inc., 1971), p. 57.

B-4. Cost of Heating a House

If the heat losses from a house can be calculated, then the cost of heating it and the savings from added insulation can be estimated. The heat losses can themselves be estimated by considering the thermal resistances of the materials used and the outside temperature.[3]

Figure IV-6 is a side view of a rectangular single-story house of width W and length L. We can estimate the heat loss through the basement walls and floor in Btuh (Btu's of heat energy per hour) from

$$\text{Rate of heat loss (basement)} = (2 \cdot W \cdot L) + [64 \cdot (W + L)]$$

where both the dimensions are to be in feet.

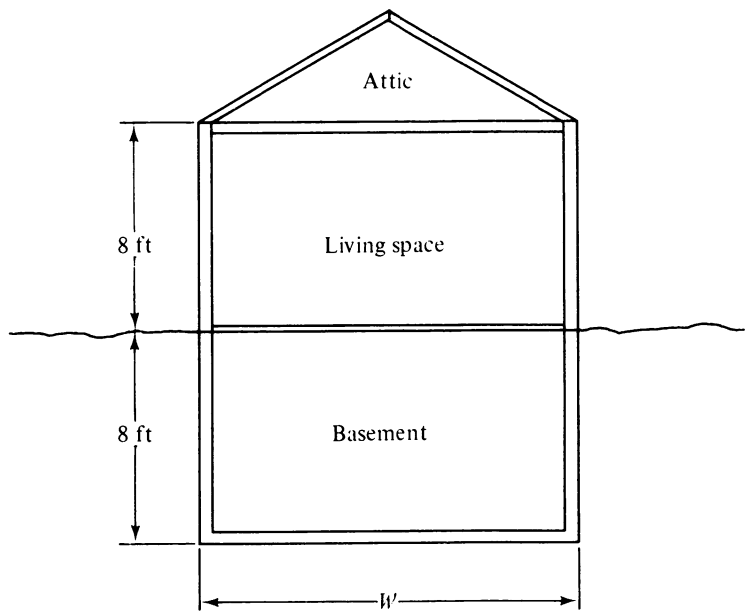


Figure IV-6.

Calculating the remaining heat losses requires some additional definitions and data. The materials making up the walls and ceiling have a certain resistance to the flow of heat through them; these thermal resistances add together to give a total resistance. Then the overall coefficient of heat transfer, U, is the reciprocal of this total resistance, and the Btuh heat loss is given by

$$\text{Rate of heat loss (ceiling and walls)} = U \cdot \text{Area} \cdot \text{Temp. diff.}$$

where *Area* is the area in square feet through which the heat is flowing and *Temp. diff.* is the temperature difference in degrees Fahrenheit between the inside and outside air. Here are

[3] For a detailed exposition of the factors involved in heat calculations, see, for example, W. J. McGuinness and B. Stein, *Mechanical and Electrical Equipment for Buildings*, 5th ed. (New York: John Wiley & Sons, 1971).

some typical thermal resistances in $ft^2 F°/Btuh$.

1. Air boundary layer (use for all solid-wall calculations): 0.85
2. Brick: 0.44
3. Concrete block: 2.00
4. Air space within wall or ceiling: 0.95
5. Plasterboard: 0.47
6. Plaster: 0.09
7. Wood sliding and sheathing: 1.89
8. Single-pane glass window: 0.88
9. Double windows: 1.79

Thus a solid 10 by 8 foot concrete block wall with an airspace and a plasterboard interior covering has a total thermal resistance of

$$2.00 + 0.95 + 0.47 = 3.42 \text{ ft}^2 \text{F°/Btuh}$$

and an overall coefficient of heat transfer of

$$U = \frac{1}{3.42} = .292 \text{ Btuh/ft}^2 \text{F°}$$

and for a temperature difference of 60 F°, it has an hourly heat loss given by

$$\text{Rate of heat loss} = (.292 \text{ Btuh/ft}^2 \text{F°})\ (80 \text{ ft}^2)\ (60 \text{ F°}) = 1.40 \times 10^3 \text{ Btuh}$$

Also, the thermal resistance of a wall or ceiling is increased by about 3.09 for *each* inch of insulation that is added.

The window area must be subtracted from the wall area to get the solid-wall area; the two areas must be considered separately in calculating the rate of heat loss. The heat loss through the ceiling can be obtained by estimating the attic temperature as being halfway between the indoor and outdoor temperatures and by using an overall coefficient of heat transfer appropriate to the materials used there.

The daily cost of heating the house is determined then by multiplying the total Btuh by 24 hours and determining the cost of fuel to provide this amount of heat energy. Natural gas costs about \$.002 for one cubic foot, and a cubic foot provides about 1050 Btu, but only about 790 Btu of this stays inside the house. Electricity costs about \$0.04 per kilowatt-hour, a kilowatt-hour is about 3410 Btu, and the transformation from electrical to heat energy is nearly 100% efficient.

Now you should be able to write a program that determines daily heating costs using various building materials and with varying amounts of insulation and with gas or electric heat. Here are three sample houses:

House #1

Size: 24 ft by 30 ft
Walls: Brick, concrete block, air space, and plasterboard
Windows: Single-pane glass, 40 sq ft in surface area
Ceiling: Plasterboard, air space, and 2 in. of insulation
Temperature difference: 70°F
Heating: Gas

Answers: Rate of heat loss (a) through basement: 4896 Btuh
(b) through solid walls: 12246 Btuh
(c) through windows: 3182 Btuh
(d) through ceiling: 3316 Btuh
Daily heating cost: $1.44

House #2

Size: Same as House #1
Walls: Same as House #1 except they contain 6 in. of insulation
Ceiling: Same as House #1 except it contains 10 in. of insulation
Temperature difference: Same as House #1
Heating: Same as House #1

Answers: Total rate of heat loss: 9721 Btuh
Daily heating cost: $0.59

House #3

Size: 30 ft by 48 ft
Walls: Wood siding and sheathing, 4 in. of insulation, air space, plaster
Ceiling: Plaster, air space, 10 in. of insulation
Temperature difference: 95°F
Heating: Electric

Answers: Total rate of heat loss: 19719 Btuh
Daily heating cost: $ 5.55

B-5. Matching Torsional Strengths

Suppose that a solid steel shaft, 2 in. in diameter, is to be fitted and welded into a hollow steel shaft having an *inside* diameter of two inches, as shown in Fig. IV-7. What must the outside diameter of the hollow shaft be if they are to have equal strength in twist (torsion)?

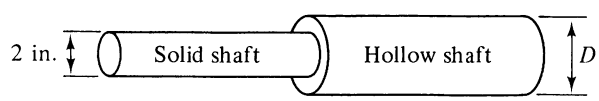


Figure IV-7.

Because the shafts are connected, they have the same torque acting on them at any one time. From the theory of linearly elastic materials, this torque is given by

$$\text{Torque} = \frac{2 \cdot \gamma \cdot J}{D}$$

where γ is the shear stress, J is the polar moment of inertia of the shaft, and D is the diameter. The polar moment of inertia for the solid shaft is

$$J_S = \frac{\pi \cdot 2^4}{32}$$

and for the hollow shaft it is

$$J_H = \frac{\pi \cdot (D^4 - 2^4)}{32}$$

Using these expressions and equating the torques for the two shafts gives us

$$\frac{\gamma \cdot (\pi/32) \cdot 2^4}{1} = \frac{\gamma \cdot (\pi/32) \cdot (D^4 - 2^4)}{D/2}$$

or

$$8 \cdot D = D^4 - 16$$

or

$$D^4 - (8 \cdot D) - 16 = 0$$

We don't need a mathematical existence theorem at this point! We know physically that the diameter must be somewhat greater than 2 and so we need to merely look for the first root that is greater than 2. Note that substituting D equal to 2 in the equation gives us -16, whereas substituting 2.5 yields +3; thus, because the function is continuous, the root we seek must be somewhere between 2 and 2.5.

One way to solve this is to ask the computer to start with D equal to 2, add small increments to D until the expression becomes positive, and then print out the previous value of D; if you then reduce the increment by a factor of ten, you can get one more significant figure in the answer, and this can be continued up to the limitations of machine precision. A more sophisticated approach is to use the root-solving algorithm given in Sec. A of Chap. III. With either method you should find a value of 2.44 in. for D.

B-6. Pipeline Droop

Most structural materials show a linear relationship between stress and strain over the range of stress values that is permitted them. This allows us to define an elastic modulus that is the constant ratio of stress to strain over this range. Then the elastic curve equation[4]

$$\frac{d^2 Y}{dX^2} = \frac{M}{EI}$$

can be used to determine the deflections of a beam whose elastic modulus is E, whose moment of inertia about the centroidal axis is I, and that is resisting a bending moment M.

[4] Gerner A. Olsen, *Elements of Mechanics of Materials*, 2nd ed. (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1966), p. 250.

One of the structural materials for which this does not work is iron. The following is a table of strain versus stress for a specimen of iron:

Strain, ϵ	*Stress, σ (thousands of lb/in.^{2})*
0.000	0.0
0.001	9.5
0.002	17.2
0.003	23.0
0.004	27.7
0.005	32.0
0.006	36.0
0.007	39.5
0.008	42.5
0.009	45.0
0.010	48.0
0.011	51.0
0.012	53.0
0.013	55.0
0.014	57.0
0.015	58.5
0.016	59.5
0.017	60.5
0.018	61.5
0.019	62.5
0.020	63.0

The fitting procedure given in Sec. C. of Chap. III allows you to fit this data to a polynomial that can then be manipulated quite easily. You should find that a fifth-degree polynomial fits the data with a chi of about 1.6×10^{-4}, and this is a fit as good as the data. Then the elastic curve equation can be rewritten in the form

$$\frac{d^2 Y}{dX^2} = \frac{\epsilon}{C}$$

where Y is the vertical coordinate for the center of the beam, X is the horizontal coordinate along the beam (which is assumed to have a symmetrical cross section), and C is the distance from the neutral axis (which is at the center of the beam if symmetrical) to the farthest point of the cross section.

Consider then the special case of a loaded iron pipe of length L that is filled with some liquid and is supported only at one end (Fig. IV-8). The moment of inertia in in.^{4} of the circular cross section of a pipe is given by

$$I = \frac{\pi \cdot (D0^4 - D1^4)}{64} = \frac{\pi \cdot (D0 - D1) \cdot (D0 + D1) \cdot (D0^2 + D1^2)}{64}$$

where $D0$ is the outer diameter in inches and $D1$ is the inner diameter in inches. (The second form of the right-hand side is more accurate if $D1$ is nearly equal to $D0$.) Also, the distance C is simply

$$C = \frac{D0}{2}$$

for such a pipe.

The bending moment at any point X from the supported end is given by

$$M = \left\{\frac{W1 + W2}{2} \cdot [L^2 - (2 \cdot L \cdot X) + X^2)]\right\} + [P \cdot (L - X)]$$

where P is the force loading the unsupported end, $W1$ is the weight per unit length of the pipe along its length, and $W2$ is the weight per unit length of the fluid in the pipe. Iron weighs about 0.282 lb/in.^{3} so

$$W1 = 0.282 \cdot \frac{\pi \cdot (D0^2 - D1^2)}{4} = 0.282 \cdot \frac{\pi \cdot (D0 - D1) \cdot (D0 + D1)}{4}$$

because the second factor is the cross-sectional area of the pipe. ($W1$ is in lb/in. when diameters are in inches.) Similarly, oil weighs about 0.0324 lb/in.^{3} If oil is the fluid, then

$$W2 = .0324 \cdot \frac{\pi \cdot D1^2}{4}$$

Finally, the stress on the pipe at any point X can be obtained from the flexure formula:

$$\sigma = \frac{M \cdot C}{I}$$

The procedure to be used in solving the differential equation is to determine the stress from this last formula, divide this by 1000 so that the strain ϵ can be determined from the fitted polynomial, and then use this ϵ to solve the equation. (See Sec. G of Chap. III.)

Figure IV-8 is drawn for a 40-foot pipe (L = 480 inches) of outer diameter 6.5 inches, of inner diameter 5 inches, and a load P equal to 50 pounds. The pipe droops 71 inches, and the angle θ in the figure is equal to 11.37°. (Note that Y and dY/dX are zero at X equal to zero.)

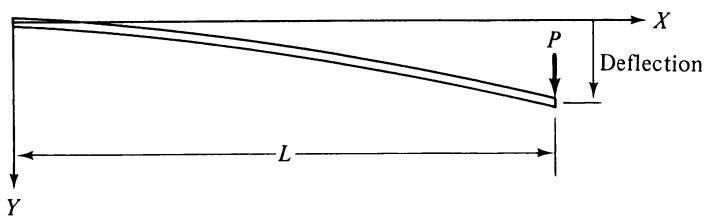


Figure IV-8.

C. Aeronautical

C-1. True Airspeed Calculation

The fact that an aircraft airspeed indicator usually gives neither the true speed relative to the air nor the speed relative to the ground is a never-ending source of confusion. Instead it measures the difference in pressure between the static air pressure and the impact air

pressure in a tube pointed into the airstream; at moderate speed this pressure difference is equal to $\frac{1}{2}\rho V^2$, where ρ is the density of the air and V is the desired airspeed. But, rather than calibrating the instrument in terms of pressure, a density equal to sea-level standard density ($\rho_0 = 2.377 \times 10^{-3}$ sl/ft^3) is substituted so that the dial can be marked in terms of speed. This means that the true airspeed must be calculated from the indicated airspeed by the formula

$$V_{TAS} = \sqrt{\frac{\rho_0}{\rho}} \cdot V_{CAS} \tag{IV-53}$$

Here ρ is the actual air density where the aircraft is flying, and (except for instrument and installation errors) the calibrated airspeed, V_{CAS}, is equal to the indicated airspeed at speeds well below the speed of sound. Note that only when the air density happens to be equal to the sea-level standard density is the true airspeed equal to the calibrated airspeed.

On a "standard" day, the density variation with altitude is (by definition) given by[5]

$$\rho = \rho_0 \cdot e^X$$

where

$$X = -\frac{H}{23111} + 0.294 \cdot \sin\frac{H}{28860} + 0.213 \cdot \sin\frac{H}{86580}$$

so that

$$V_{TAS} = e^{-X/2} \cdot V_{CAS}$$

In these formulas H is the altitude in feet, but the speed can be in any appropriate unit.

So it could be helpful to a pilot if you calculated the true airspeed for a calibrated airspeed of 120 mi/hr at a number of altitudes, say from sea level through 20,000 ft in steps of 2000 ft. Some representative answers are $H = 0$ ft, $V_{TAS} = 120$ mi/hr; $H = 10{,}000$ ft, $V_{TAS} = 140.00$ mi/hr; and $H = 20{,}000$ ft, $V_{TAS} = 164.333$ mi/hr. These kinds of numbers are put in aircraft owner's manuals.

Unfortunately, the weather is never very standard. And so the pilot, by use of the ideal gas law, must calculate the actual air density from the calculated air pressure (using the altimeter) and from the measured air temperature. From the indicated altitude the pilot can calculate the pressure in lb/ft^2 from

$$P = 2116.22 \cdot \exp(-3.47 \times 10^{-5} \cdot H)$$

and then obtain the actual air density in sl/ft^3 from

$$\rho = \frac{(5.82585 \times 10^{-4}) \cdot P}{(T + 459.688)}$$

[5] See Daniel O. Dommasch, Sydney S. Sherby, and Thomas F. Connolly, *Airplane Aerodynamics*, 4th ed. (New York: Pitman Publishing Corporation, 1967), for this and succeeding formulas.

where T is the measured temperature in degrees Fahrenheit. The standard temperature for a given altitude, on the other hand, is given by

$$T = 518.688 \cdot [1 - (6.875 \times 10^{-6} \cdot H)] - 459.688$$

To determine the influence of this nonstandard air density, a nice calculation would be one that determined the true airspeed at 10,000 ft for temperatures 20 and 40 Fahrenheit degrees above and below standard. For a calibrated airspeed of 120 mi/hr, I find (using Eq. IV-53) a true airspeed of 131.918 mi/hr for 40°F below standard and a true airspeed of 143.334 mi/hr for 40°F above standard.

C-2. Wind Drift and Ground Speed

Student pilots on their first cross-country flight go through all sorts of mental and physical contortions. What are they afraid of? Primarily they are trying to avoid getting lost! And the enemy (besides themselves) is the wind.

The problem is one of properly adding two vectors, the velocity of the airplane relative to the air and the velocity of the air relative to the ground, to find the resulting velocity of the airplane relative to the ground, as shown in Fig. IV-9. It would be a much simpler problem if the pilot had already decided which way he was going to point the plane; instead, he usually has a destination and a direction over the ground already picked out, and he wants

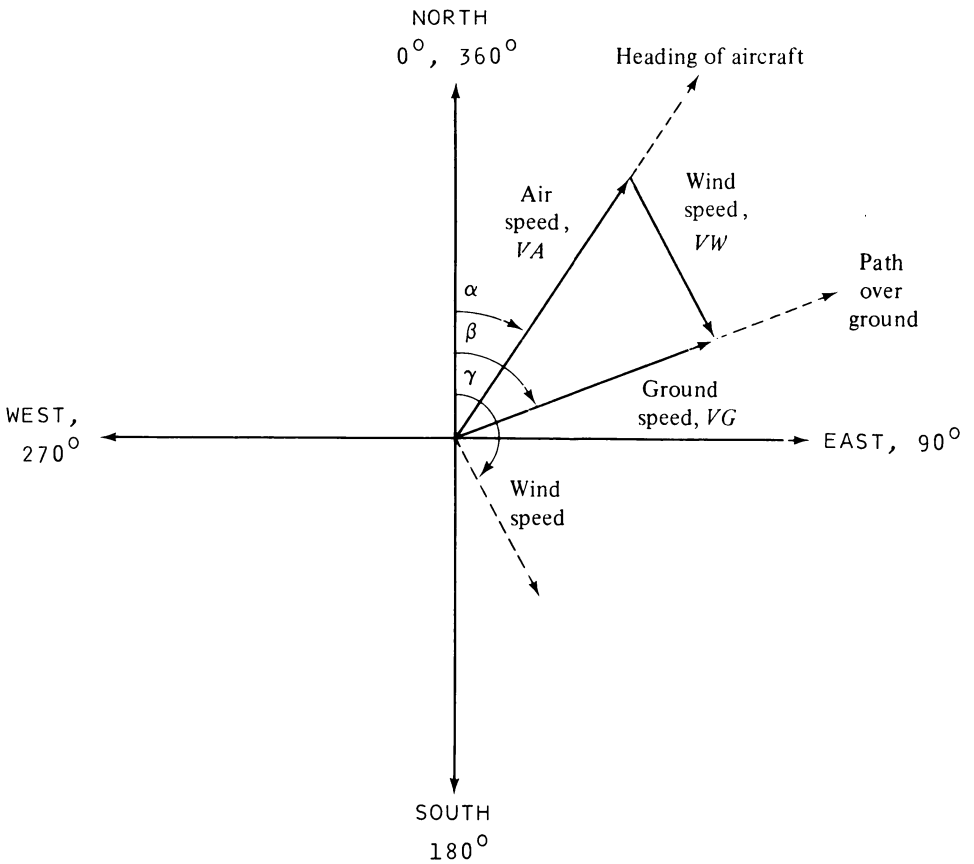


Figure IV-9.

to *determine* the direction to point the machine and the resulting ground speed. For light aircraft the ground-speed calculation is often the more important; the pilot can often use navigational radio to stay on course, but in-flight refueling isn't available for the pilot who realizes too late that cars on the ground are going faster than he is! (I'm using the pronoun *he* for convenience, but I mean both male and female aviators; assuredly, their problems with the wind are identical.)

In Fig. IV-9 all the velocity directions are measured relative to magnetic north. The angle α is the required airplane heading for a wind blowing *in* the direction given by γ and for a destination in the direction given by β. The magnitude of the wind speed is denoted by VW; the aircraft's cruising speed is VA, and the resulting speed over the ground is VG. The vector addition can be accomplished analytically with the help of the law of sines and the law of cosines; the result is

$$\alpha = \beta - \arctan \left[\frac{X}{1 - X^2} \right]^{1/2}$$

where

$$X = \frac{VW}{VA} \cdot \sin(\gamma - \beta)$$

and

$$VG = \{[VA^2 + VW^2 + [2 \cdot VA \cdot VW \cdot \cos(\gamma - \alpha)]\}^{1/2}$$

Here are some sample numbers to check your program; note that the first two are simple cases for which you can guess the answer beforehand.

	VA (mph)	*β (deg)*	*VW (mph)*	*γ (deg)*	*Answers:*	*α (deg)*	*VG (mph)*
1.	100	90	0	0		90	100
2.	110	45	50	225		45	60
3.	110	45	40	90		30.1002	134.586
4.	643	32.3	45	50		31.0808	685.724
5.	98	336	100	182.34		2.92038	2.23738
6.	70	137	80	90		193.705	92.98888

I bet you didn't spot the bug! In sample 5 the wind is actually too strong for it to be possible for the aircraft to go in the intended direction. But the computer calmly returns a recommended heading that sends the aircraft on a path over the ground of 156° rather than the desired 336°! Try that for a bug in your program if it is directing a slow-speed guided missile! Can you figure out how to detect and eradicate the pest?

C-3. Collision Avoidance Radar

The U.S. is nearly covered with radar sets that are teamed with computers to look ahead for potential airspace conflicts. The computer uses the current positions and speeds of all the aircraft in its area of responsibility and computes their future positions to see if they will get too close together, assuming that their speeds don't change. This calculation is constantly being updated to include newly arrived aircraft and changing speeds.

You can begin to appreciate what is involved by simulating these calculations for the 500 by 1000 mile block of airspace shown in Fig. IV-10. Victor 66 is an east airway, and Victor 41 is a north airway; they cross precisely in the middle of the block. Positions are measured from the left edge for aircraft on V-66 and from the bottom edge for aircraft on V-41. Speeds should all be positive; a negative speed means that an aircraft is going the wrong way on the airway. No calculations are necessary for this latter case, but a warning message should be printed out immediately, giving the aircraft and its present position. There are two other hazardous possibilities for which you must check and report: (a) Two aircraft on the same airway which get within 6 miles of each other before they both pass off the airspace block, and (b) two aircraft on different airways which get within twelve miles of each other. In each case, the aircraft involved, the time of the impending conflict, and the aircraft locations at that time should be reported.

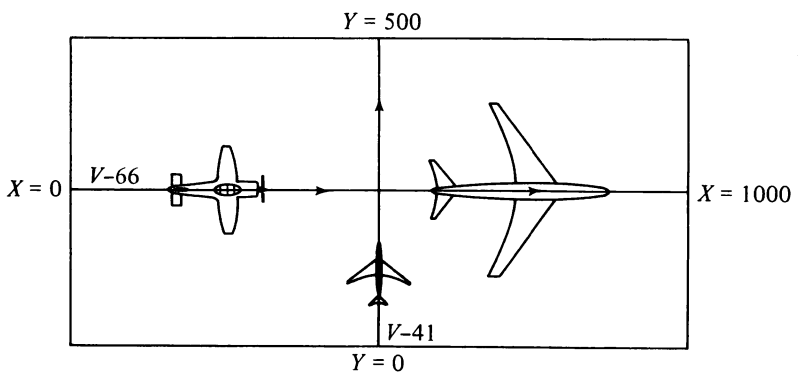


Figure IV-10.

For the first check, suppose that the speed and position of the faster aircraft are $V1$ and $X1$, respectively, and for the slower aircraft are $V2$ and $X2$, respectively. Then the check must be made only if the faster aircraft is behind the slower one—that is, only if $X1$ is less than $X2$. The position of the faster aircraft at a later time T is given by

$$D1 = X1 + (V1 \cdot T)$$

and for the slower aircraft is given by

$$D2 = X2 + (V2 \cdot T)$$

and therefore the time $T1$ when they get within 6 miles of each other is obtained by solving the equation

$$D1 + 6 = D2$$

to obtain

$$T1 = \frac{(X2 - X1 - 6)}{(V1 - V2)}$$

The positions $D1$ and $D2$ should be checked for this time to see if both aircraft are off the airspace block and, if so, no message is printed.

The second check is more difficult. One efficient method is to see which aircraft will first approach to within 9 miles of the intersection, then use small increments of time, and calculate the separation distance after each increment; this should be discontinued when this first aircraft has gone 12 miles past the intersection and the possibility for a conflict no longer exists. An efficient time increment ΔT for this process would be one that advances the faster aircraft 0.5 mile each time; this increment is given by

$$\Delta T = \frac{1}{2 \cdot V}$$

where V is the speed of the faster aircraft, and the time increment is in fractions of an hour if the speed is in miles per hour.

If the aircraft on the east airway has a position and speed given by $X1$ and $V1$, then it will get to within 9 miles of the intersection at a time $T2$ given by

$$T2 = \frac{491 - X1}{V1}$$

Similarly, if the aircraft on the north airway has a position and speed given by $Y1$ and $V2$, then it will get to within 9 miles of the intersection at a time $T3$ given by

$$T3 = \frac{241 - Y1}{V2}$$

The separation distance between two aircraft on separate airways can be calculated from

$$\text{Separation distance} = \sqrt{(500 - X1)^2 + (250 - Y1)^2}$$

where $X1$ and $Y1$ are their instantaneous positions on V-66 and V-41, respectively.

Considering all the logical decisions to be made in this program, a flowchart is the logical place to start. Assume that the information about each aircraft is given by a sequence of four numbers. The first number is either 66 or 41 for the airway it is on, the second number is the flight number of the aircraft, the third number is the current position of the aircraft, and the last number is its speed. If the first number is something besides 66 or 41, it means that the list of aircraft has been completed. Write your program in generality for any number of aircraft up to six on each airway, and print out the time and positions and aircraft numbers for each possible future conflict.

Try the data 66, 1, 500, 70; 66, 2, 400, 500 to check V-66; you should find a conflict after .218605 hours. Try the data 41, 3, 200, 100; 41, 4, 180, 300 to check V-41; you should find a conflict after .07 hours. Put all this data into the same program run, and you should find an airway conflict between Flights 2 and 4 after .204333 hours.

Finally, to really keep the controllers hopping, try this data!

66, 1, 325, 100; 66, 2, 800, 275; 66, 3, 500, 125; 66, 6, 100, 108;
66, 5, 450, 150; 66, 6, 750, 800; 41, 7, 15, 250; 41, 8, 163, 200;
41, 9, 200, 175; 41, 10, 255, 400; 41, 11, 400, 600; 41, 12, 475, 70

You should find conflicts between Flights 2 and 6 and between Flights 3 and 5 on V-66; between Flights 8 and 9 and between Flights 11 and 12 on V-41; and airway conflicts between Flights 3 and 10 and between Flights 5 and 9.

C-4. Turns About a Point

Private pilots must demonstrate their ability to compensate for the wind while flying a prescribed path relative to the ground. One of the tests of this is the ability to make a constant-radius turn about a point on the ground; the angle of bank is continuously varied to correct for wind drift, and normally the pilot is expected to choose a radius that results in a maximum bank of 45°. Assuming a wind from the north, the problem can be diagrammed as in Fig. IV-11.

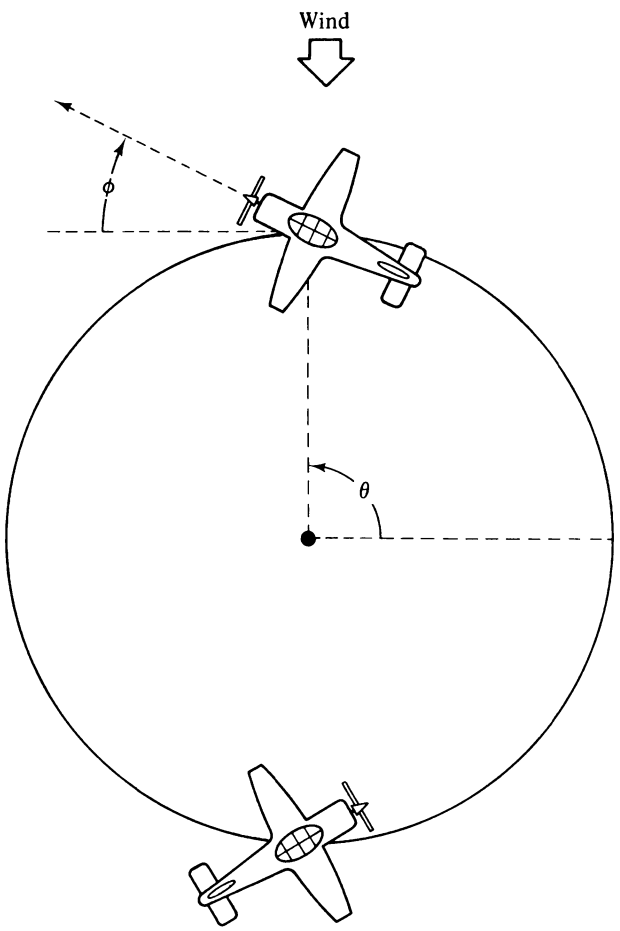


Figure IV-11.

We would like to calculate the wind correction angle, the ground speed VG, and angle of bank β, and the rate of turn at each point of the circle. Also, it is of interest to determine the total time for the maneuver. Here then are the formulas that describe the motion.

$$[VA \cdot \sin(\phi) \cdot \tan(\theta)] - [VW \cdot \tan(\theta)] - [VA \cdot \cos(\phi)] = 0 \qquad \begin{matrix} -45° \leqslant \theta \leqslant 45° \\ 135° \leqslant \theta \leqslant 225° \end{matrix} \qquad \text{(IV-54)}$$

$$[VA \cdot \sin(\phi)] - VW - [VA \cdot \cos(\phi) \cdot \cot(\theta)] = 0 \qquad \begin{matrix} 45° \leqslant \theta \leqslant 135° \\ 225° \leqslant \theta \leqslant 315° \end{matrix} \qquad \text{(IV-55)}$$

$$VG = \left([VA \cdot \cos(\phi)]^2 + \{[VA \cdot \sin(\phi)] - VW\}^2\right)^{1/2} \qquad \text{(IV-56)}$$

$$\text{Wind correction angle} = \theta + \phi - 90° \qquad \text{(IV-57)}$$

$$\tan(\beta) = \frac{VG^2}{R \cdot G} \qquad \text{(IV-58)}$$

$$\text{Rate of turn (deg/sec)} = \frac{180 \cdot G \cdot \tan(\beta)}{\pi \cdot VG} \qquad \text{(IV-59)}$$

where

β = angle of bank of aircraft

ϕ = heading of aircraft

R = radius of the turn in ft

VG = ground speed of the aircraft in ft/sec

VW = wind speed in ft/sec

VA = air speed of the aircraft in ft/sec

G = acceleration due to gravity = 32.2 ft/sec^2

From Eq. IV-58 it can be seen that the minimum bank for the aircraft will occur at $\theta = 0°$ when it is flying directly into the wind and the ground speed is a minimum ($VG = VA - VW$). The maximum bank for the aircraft will occur at $\theta = 180°$ when the ground speed is a maximum ($VG = VA + VW$).

If you haven't studied Sec. A of Chap. III on roots of equations, you can still solve Eqs. IV-54 and IV-55 by trial and error. However, if you do use the algorithm given there, you can easily make the calculations for each of the 360° of the circle. Then you can determine the time required for the circle by noting that the time in seconds for each degree of turn is very nearly equal to 1° divided by the rate of turn from Eq. IV-59.

The radius of the turn can be obtained at the beginning of your program from Eq. IV-58 by inserting the maximum desired bank for β and by using the maximum ground speed given previously. Remember to convert degrees to radians; convert mph to ft/sec by multiplying mph by (5280/3600). Here are some representative answers:

VW (mph)	*VA (mph)*	*R (ft)*	*max φ (deg)*	*min β (deg)*	*max β (deg)*	*max rate (deg/sec)*
25	70	602.913	20.925	12.6464	45	13.2411
25	110	1217.52	13.137	21.625	45	9.3178
25	160	2411.65	10.807	25.0864	45	6.62054

The total time for the first line is 40.9 seconds for the complete circle.

There is one approximation in the above calculations that can be improved. The aircraft speed actually decreases as the bank is increased because of an increase in the drag

force (assuming no loss or gain in altitude). For a popular trainer this speed variation can be estimated from

$$176288 = \left[.0367 + \frac{(8563)^2}{\pi \cdot (6.524) \cdot V^4 \cdot \cos^2(\beta)} \right] \cdot V^3 \qquad \text{(IV-60)}$$

The initial speed (at $\theta = 0$) can be obtained by making a guess at the bank angle at that point and correcting it after a more accurate speed has been found. Because it is a small effect, you can then use the previous value for the bank to calculate the new airspeed, VA, as long as the incrementing angle is a degree or less.

Here are some representative answers for this calculation: For a wind speed of 25 mph and a zero-bank aircraft speed of 110 mph, the radius needed for a maximum bank of 45° is 1107 ft. The actual aircraft speed at the minimum bank is 109.0 mph; at the maximum bank it is 103.7 mph. The maximum wind correction angle is 13.45°; the minimum angle of bank is 23.05°; the maximum rate of turn is 9.7711°/sec.; and the total time for the turn is 46.02 sec. For the same conditions, except with a maximum bank of 55°, the radius needed is 668.56 ft; the actual aircraft speed at the minimum bank is 107.1 mph; at the maximum bank it is 94.55 mph. The maximum wind correction angle is 15.03°; the minimum angle of bank is 34.1°; the maximum rate of turn is 15.03°/sec.; and the total time for the turn is 28.93 sec.

C-5. Keeping the Wings On

An aeronautical engineer has designed a sleek little two-seater aircraft (Fig. IV-12) that is to have a gross weight (W) of 1600 lb. He has determined that landing-speed requirements imply a wing area (S) of 197 ft^2. (The wing area is defined as the projected area of the wing and the area of the part of the fuselage between the wings.) And for acceptable

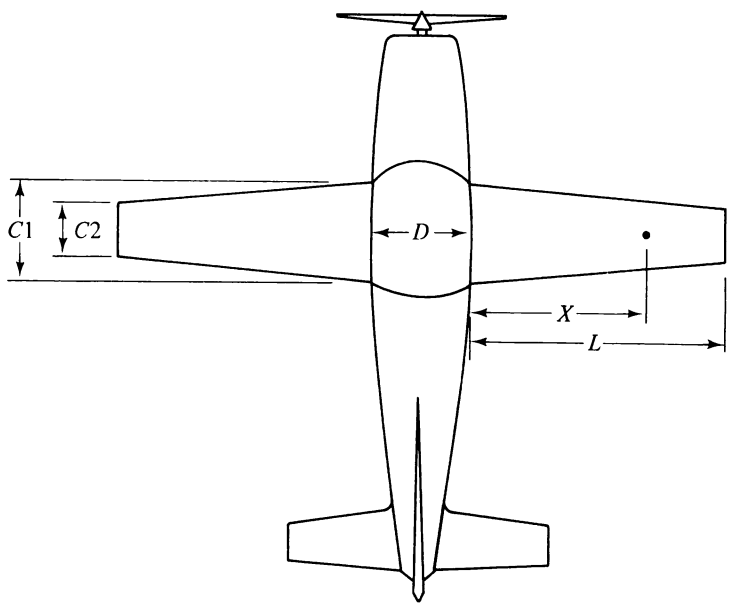


Figure IV-12.

passenger comfort, he wants the fuselage to have a width (D) of 4.92 ft. The next item is to determine the shearing stresses and the bending moments at different points along the wing for various wing shapes. He knows that a rectangular wing is the cheapest to build but that a tapered wing is more efficient aerodynamically. Furthermore, he thinks that he might want to make the aircraft strong enough for aerobatics and would like to know how much additional strength is necessary for this.

The requirements for a utility category plane are that the structure be undamaged up to a load factor (N) of 4.4; for an aerobatic category aircraft, it is 6.0. (The load factor is the ratio of the lift of the wings to the gross weight of the aircraft.) The weight of the wings ($W1$) depends on the particular category, too; for the given surface area, its value in pounds can be calculated from

$$W1 = 102 \cdot \sqrt{N}$$

The loads on the spar and the fuselage-wing attach fittings depend on the weight actually supported by the wings; if we call this weight $W3$, then it can be calculated from

$$W3 = W - W1 - W2$$

where $W2$ is the weight of the fuel carried in the wings. For this aircraft, our designer has determined that $W2$ will vary between 3 lb ($\frac{1}{2}$ gallon) and 156 lb (26 gallons).

The distance from the fuselage to the wing tip depends on the amount of taper that is chosen:

$$L = \frac{S - (D \cdot C1)}{C1 + C2}$$

where $C2$ and $C1$ are the inner and outer widths of the wing respectively.

Then the shear force at any point X between the fuselage and the wing tip is given by

$$V = \frac{N \cdot W3 \cdot (L - X)}{2 \cdot L^2 \cdot (C1 + C2)} \cdot [C1 \cdot (L - X)] + [C2 \cdot (L + X)]$$

and the bending moment at the same location is given by

$$M = \frac{-N \cdot W3 \cdot (X - L)^2}{6 \cdot L^2 \cdot (C1 + C2)} \cdot \left([(C1 - C2) \cdot X - [C1 + (2 \cdot C2) \cdot L] \right)$$

So help this man out by determining V and M at $X = 0$ and $X = L/2$ for utility and aerobatic category airplanes; do this for a rectangular wing ($C1 = 4.74$ ft, and $C2 = C1$) and a partially tapered wing ($C1 = 4.74$ ft, and $C2 = C1/2$) and for minimum and maximum fuel.

You should find, for example, that at the wing root ($X = 0$), the maximum shear force is 4041.46 lb (rectangular wing, $N = 6$, minimum fuel), whereas the maximum bending moment is 43,876.7 ft-lb (tapered wing, $N = 6$, minimum fuel). At the midpoint of the wing ($X = L/2$), the minimum shear force is 1127.54 lb (tapered wing, $N = 4.4$, full fuel), whereas the minimum bending moment is 6197.16 ft-lb (rectangular wing, $N = 4.4$, full fuel).

Our designer decides to go all the way—make the wings tapered and make it aerobatic! But now he is curious about how much the wing will be bent when it is under a load factor of 6. He thinks he'll use a spar that tapers from 6 in. height to 3 in. at the tip; the moment

of inertia for this spar, in ft^4, is given by

$$I = \frac{1}{576}\left[1 - \frac{X}{2 \cdot L}\right]$$

and the deflection can be obtained by solving the differential equation

$$\frac{M}{EI} = \frac{d^2 Y}{dX^2}$$

with the initial conditions that $X = 0$, $Y = 0$, $dY/dX = 0$. E is the modulus of elasticity and is 1.44×10^9 lb/ft^2 for aluminum. You should calculate a deflection of $Y = 2.777$ ft at the wing tip ($X = L = 24.4275$ ft)!

C-6. Time To Do a Loop

The loop is probably the most universally enjoyed of all the common aerobatic maneuvers (Fig. IV-13). By making some simplifying assumptions, we can obtain relatively simple formulas for the variation in important parameters as the loop progresses.

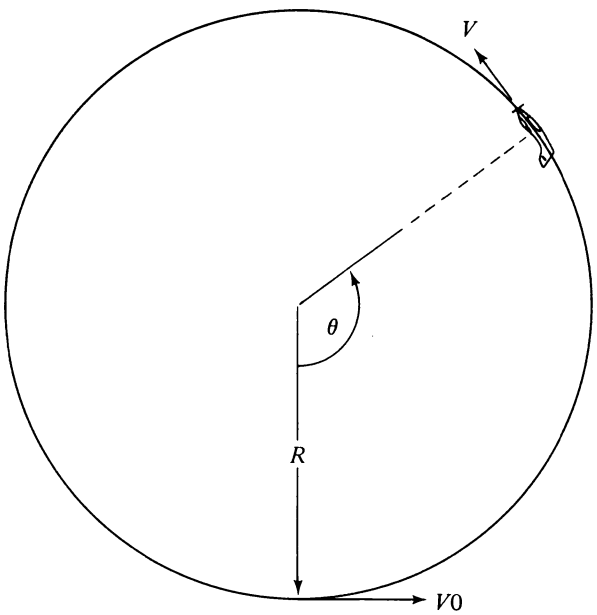


Figure IV-13.

We assume that the loop is circular and that the pilot adjusts the throttle so that the propeller thrust or jet thrust is at all times equal to the air drag on the plane. We also assume that the wings are pushing up with a force equal to one-half the weight of the plane at the very top of the loop. This is enough to completely define the motion.

Then, for an entry speed $V0$ at the bottom of the loop, we would like to determine the speed V and the execution time T as a function of the angle θ. This can be accomplished

using the principle of conservation of energy and Newton's Second Law; the results are

$$V = V0 \cdot \sqrt{\frac{2.5 + (2 \cdot \cos\theta)}{4.5}}$$

$$T = \frac{V0}{\sqrt{9} \cdot G} \int_0^{\theta} (1.25 + \cos\theta)^{-1/2} \, d\theta$$

where G is the acceleration due to gravity, and the integral, giving the time from the bottom of the loop to any angle θ, is easily evaluated with Simpson's Rule (Sec. E of Chap. III).

Another parameter of interest is the loading on the wings; the flight load factor is defined as the ratio of the lift by the wings to the weight of the plane and is given by

$$N = 2.5 + (3 \cdot \cos\theta)$$

Also, the radius of this perfectly circular loop can be calculated from

$$R = \frac{V0^2}{4.5 \cdot G}$$

The Cessna 150 Aerobat has a suggested loop entry speed of 130 mi/hr. Try using this entry speed with a G of 32.2 ft/sec^2 to calculate and print out a table of θ, V, and T in 30° increments all the way around the loop. You'll have to convert the entry speed to ft/sec (multiply by 5280/3600) for the integral equation and the radius equation but not for the speed equation. You should find, for example, that the minimum speed is 43.3333 mi/hr at the top, the load factor varies from +5.5 (entry) to −0.5 (top) to +5.5 (bottom), the total time is 13.309 seconds, and the loop radius is 250.889 ft.

Incidentally, experimental values for some loops of questionable symmetry are a maximum load factor of about 3, an average time of about 17 seconds, and an average radius of about 270 ft.

C-7. Performance Tables

The owner's manual for an aircraft always includes a table of cruising speeds versus altitude and power. It is possible to calculate the performance at high altitudes from experimental data at lower altitudes. To do this, we need to determine the drag coefficient CD as a function of lift coefficient CL. The drag coefficient can be obtained from

$$CD = \frac{1100 \cdot P}{\rho \cdot V^3 \cdot S} \qquad \text{(IV-61)}$$

where P is the horsepower required to give a speed V in ft/sec at an altitude where the air density is ρ, for an aircraft with a wing surface area S in square feet. The lift coefficient can be obtained from

$$CL = \frac{2 \cdot W}{\rho \cdot V^2 \cdot S} \qquad \text{(IV-62)}$$

where W is the weight of the aircraft in pounds. (Ordinarily the propeller efficiency would have to be included in the equation for CD, but we use constant-rpm data, so it is a constant that can be absorbed into CD.)

This performance table is taken from the owner's manual for a light single-engine aircraft:

Altitude (ft)	*Percent Power*	*Speed (mph)*	*Air Density (sl/ft^3)*
2500	76	158	.002202
	72	155	
	68	151	
	64	148	
5000	78	163	.002048
	74	160	
	70	157	
	66	153	
7500	69	159	.001887
	65	154	
	60	150	
	56	145	

For this aircraft the gross weight W is 3600 lb, and the wing surface area S is 174 square feet. The power in horsepower is obtained by multiplying the percent power in the table by 2.85.

If you take the data from 2500 ft and 5000 ft, you should find that a fit to the equation

$$CD = A0 + (A1 \cdot CL) + (A2 \cdot CL^2) \tag{IV-63}$$

yields $A0 = .0580261$, $A1 = -.0658799$, and $A2 = .122767$. (See Sec. C. of Chap. III.) For this calculation the speeds have been converted to ft/sec by multiplying the speeds in mph by 5280/3600.

Then the speeds for higher altitudes can be obtained by finding the speed that satisfies both equations for CD. This requirement can be written as

$$\frac{1100 \cdot P}{V^3 \cdot S} - \left\{A0 + \left[A1 \cdot \left(\frac{2 \cdot W}{\rho \cdot S}\right) \cdot \left(\frac{1}{V^2}\right)\right] + \left[A2 \cdot \left(\frac{2 \cdot W}{\rho \cdot S}\right)^2 \cdot \left(\frac{1}{V^4}\right)\right]\right\} = 0 \tag{IV-64}$$

and the root-solving routine of Sec. A, Chap. III, can be used to calculate the speeds for an altitude of 7500 feet. You should calculate speeds within about 1 mph of those given in the table.

D. Electronics

D-1. Design of T-Section Bandpass Filter

Many types of active and passive filters can be designed quickly with the help of computer programs. A simple example is the T-section filter[6] made up of inductances and capacitances, as depicted in Fig. IV-14. This filter tends to pass only those frequencies in the band between $F1$ and $F2$ to the load resistor R. The component values are calculated from

[6] Hugh Hildreth Skilling, *Electrical Engineering Circuits* (New York: John Wiley & Sons, Inc., 1957), p. 621.

these formulas:

$$L1 = \frac{R}{2 \cdot \pi \cdot (F2 - F1)}$$

$$L2 = \frac{R \cdot (F2 - F1)}{4 \cdot \pi \cdot F1 \cdot F2}$$

$$C1 = \frac{F2 - F1}{2 \cdot \pi \cdot F1 \cdot F2 \cdot R}$$

$$C2 = -\frac{1}{\pi \cdot (F2 - F1) \cdot R}$$

Suppose that the desired band is from $F1 = 2000$ Hz to $F2 = 5000$ Hz into a load resistance of 300 ohms. What component values should be used? (*Answers*: $L1 = 1.59155 \times 10^{-2}$ H, $L2 = 7.16197 \times 10^{-3}$ H, $C1 = 1.59155 \times 10^{-7}$ F, $C2 = 3.53678 \times 10^{-7}$ F.)

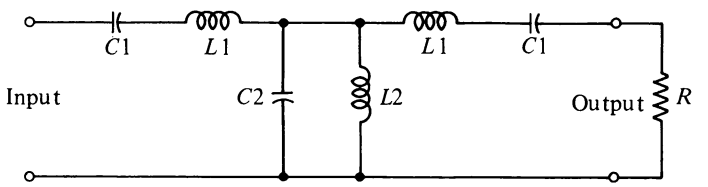


Figure IV-14.

D-2. CMOS RC Oscillator

A low-cost, low-power square wave oscillator with reasonably good stability can be built using a single CMOS IC (complementary metal-oxide semiconductor integrated circuit), two resistors, and a capacitor. The IC requires only about 1×10^{-8} watts, and the resistors and capacitor are much less expensive than the crystal that is normally used. The circuit is shown in Fig. IV-15; the frequency in Hz of the oscillator is approximately equal to that given by the expression[7]

$$F = \frac{1}{2 \cdot R1 \cdot C \cdot [(0.405 \cdot R2)/(R1 + R2 + .693]}$$

Suppose you have the IC and *one* each of the following circuit elements: resistors (values are in ohms)—47, 2200, 10×10^3, 47×10^3, 1×10^6; capacitors (values are in farads) —4.7×10^{-6}, 1×10^{-6}, $.22 \times 10^{-6}$, $.047 \times 10^{-6}$. What are the lowest and highest frequency oscillators that you can build? *Answers*: .14958 Hz and 207747 Hz, respectively. (*Hint*: The algorithm of Fig. II-7 will save paper.)

This result points out the possibility of a simple, inexpensive digital stopwatch, because a frequency of 10 Hz, giving a least count of 0.1 second, falls within the calculated range. By printing out all the possible frequencies in the foregoing calculation, you can determine that the 4.7×10^{-6} farad capacitor teamed with an $R1$ equal to 10×10^3 ohms and an $R2$

[7]Mike Watts, *CMOS Integrated Circuits*, National Semiconductor Corporation catalog, (March 1975), p. 227.

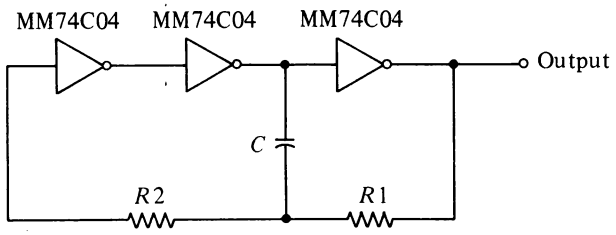


Figure IV-15.

between 47×10^3 ohms and 1×10^6 ohms will do the job. The equation can be solved for $R2$ by simple algebra, and it can be determined that the required $R2$ is 108,500 ohms.

But now another possible problem occurs to you. Will the stopwatch retain its laboratory accuracy when subjected to cold football games or hot auto races? You find that a good metal film resistor has a temperature coefficient of 1.5×10^{-4} per degree Celsius, so its resistance at any Celsius temperature can be calculated from

$$R = R0 \cdot \left\{1 + [1.5 \times 10^{-4} \cdot (T - 20)]\right\}$$

where $R0$ is the (rated) resistance at 20°C. Also you find that the solid tantulum capacitor you were planning to use gains 1 percent in capacitance at 50°C and loses 2 percent at −30°C. Try determining how much the frequency will change from these effects. You should find that it will decrease to 9.85666 Hz at 50°C (122°F) and increase to 10.2812 Hz at −30°C (−22°F). Better plan on using a trimmer resistor!

D-3. DC Current Loops

Circuits containing a current source in each loop must be analyzed using Kirchhoff's laws; one equation is obtained for each loop, and the resulting equations must be solved simultaneously. If the circuit shown in Fig. IV-16 is so analyzed, the resulting equations are

$$\begin{aligned}
[(R1 + R2 + R3) \cdot I1] - (R3 \cdot I2) + (R2 \cdot I3) &= E1 \\
(-R3 \cdot I1) + [(R3 + R4 + R5) \cdot I2] + (R5 \cdot I3) &= E2 - E3 \\
(R2 \cdot I1) + (R5 \cdot I2) + [(R2 + R5 + R6) \cdot I3] &= E1 - E3
\end{aligned}$$

This is a good opportunity to use the algorithm described in Sec. B of Chap. III. Try the following circuit values: $R1 = 7.5\ \Omega$, $R2 = 3.9\ \Omega$, $R3 = 4.3\ \Omega$, $R4 = 11\ \Omega$, $R5 = 3.3\ \Omega$, $R6 = 2.7\ \Omega$, $E1 = 15\ V$, $E2 = 12.6\ V$, and $E3 = 4.5\ V$. You should find that the current $I1$ has a value of .99709 A, $I2$ a value of .581926 A, and $I3$ a value of .473838 A.

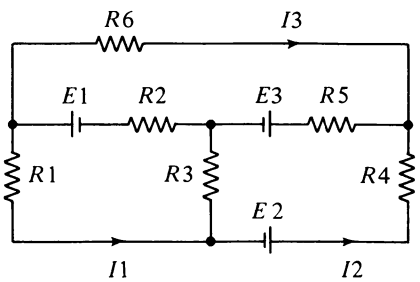


Figure IV-16.

D-4. Electron Travel Time

A basic geometry for a rectifier tube is shown in a top view in Fig. IV-17; the inner cylindrical tube is the *cathode*, and the electrons travel from there to the outer cylindrical tube, the *anode*. For those electrons that obtain all their energy from the accelerating voltage V, the time for them to travel the distance is still difficult to calculate, because the electric field, and therefore the acceleration of the electrons, is not constant in the inter-electrode space. However, the time in seconds is *approximately* given by[8]

$$T = \left(\frac{2 \cdot M}{Q \cdot V}\right)^{1/2} \cdot R1 \cdot Z \cdot \left(1 + \frac{Z}{3} + \frac{Z^2}{10} + \frac{Z^3}{42} + \frac{Z^4}{216}\right)$$

where

Q = charge of the electron = 1.60206×10^{-19} coulombs

M = mass of the electron = 9.1083×10^{-31} Kg

V = accelerating voltage in volts

$R1$ = radius of the inner electrode in cm

$R2$ = radius of the outer electrode in cm

$Z \equiv \ln (R2/R1)$

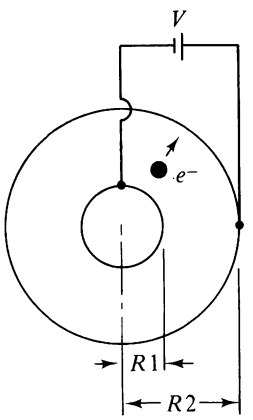


Figure IV-17.

For an accelerating voltage of 300 volts, an inner radius of 0.1 cm, and an outer radius of 1.0 cm, you should find a transit time of 1.21866×10^{-7} seconds. (*Hint*: You can improve the efficiency of this calculation with factoring.)

[8] George F. Corcoran and Henry W. Price, *Electronics* (New York: John Wiley & Sons, Inc., 1954), p. 42.

There still exists the possibility of a more accurate calculation: You can use Simpson's Rule (Sec. E of Chap. III) to solve

$$T = R1 \cdot \left(\frac{2 \cdot M \cdot Z}{Q \cdot V}\right)^{1/2} \cdot \left[(Z^{1/2} \cdot e^{Z}) - \int_0^z (X^{1/2} \cdot e^{X} \cdot dX)\right]$$

From this formula you should find a time of 1.2505×10^{-7} seconds.

D-5. Power Supply with Capacitive Filtering

Figure IV-18a shows an ac generator supplying power to a load resistor R through a diode and with a capacitor smoothing out the voltage fluctuations across the resistor.[9] The generator voltage is assumed to have a sinusoidal variation with an angular frequency ω and with maximum voltages of $\pm E$; two cycles are shown in Fig. IV-18b. If the diode has a forward resistance much less than R but a backward resistance much greater than R, then the voltage across R in the *absence* of the capacitor is that shown in Fig. IV-18c. When the capacitor is inserted across R, as shown, it charges to the maximum positive voltage with the voltage across R, but when the supply voltage starts to decrease, it discharges through R, thereby preventing the voltage across R from dropping to zero. The result is that the voltage across R has an average value $E1$ (the dc voltage) but also has significant fluctuations (ripple) about that value, as shown in Fig. IV-18d. This circuit is a half-wave rectifier dc power supply and is usable for small output currents or when relatively large ripple in the output voltage can be tolerated. We wish to determine the time during the cycle that the diode is conducting ($\alpha 1$ to $\alpha 2$), the value of $E1$, and a measure of the ripple. We define the ripple factor as

$$RF = \frac{\sqrt{\langle E^2 \rangle - E1^2}}{E1}$$

where $\langle E^2 \rangle$ is the average value over a cycle of the *square* of the voltage across R.

If we make the definitions

$$Q = \omega \cdot C \cdot R$$

$$M = \frac{2 \cdot Q^3}{1 + Q^2}$$

$$N = \frac{Q^2}{(1 + Q^2)^{1/2}}$$

$$\alpha = \alpha 2 - \alpha 1$$

$$S = \exp\left\{-\frac{[(2 \cdot \pi) - \alpha]}{Q}\right\}$$

then after we calculate $\alpha 2$ from

$$\alpha 2 = \arctan(-Q)$$

[9] George F. Corcoran and Henry W. Price, *Electronics* (New York: John Wiley & Sons, Inc., 1954), pp. 78–85.

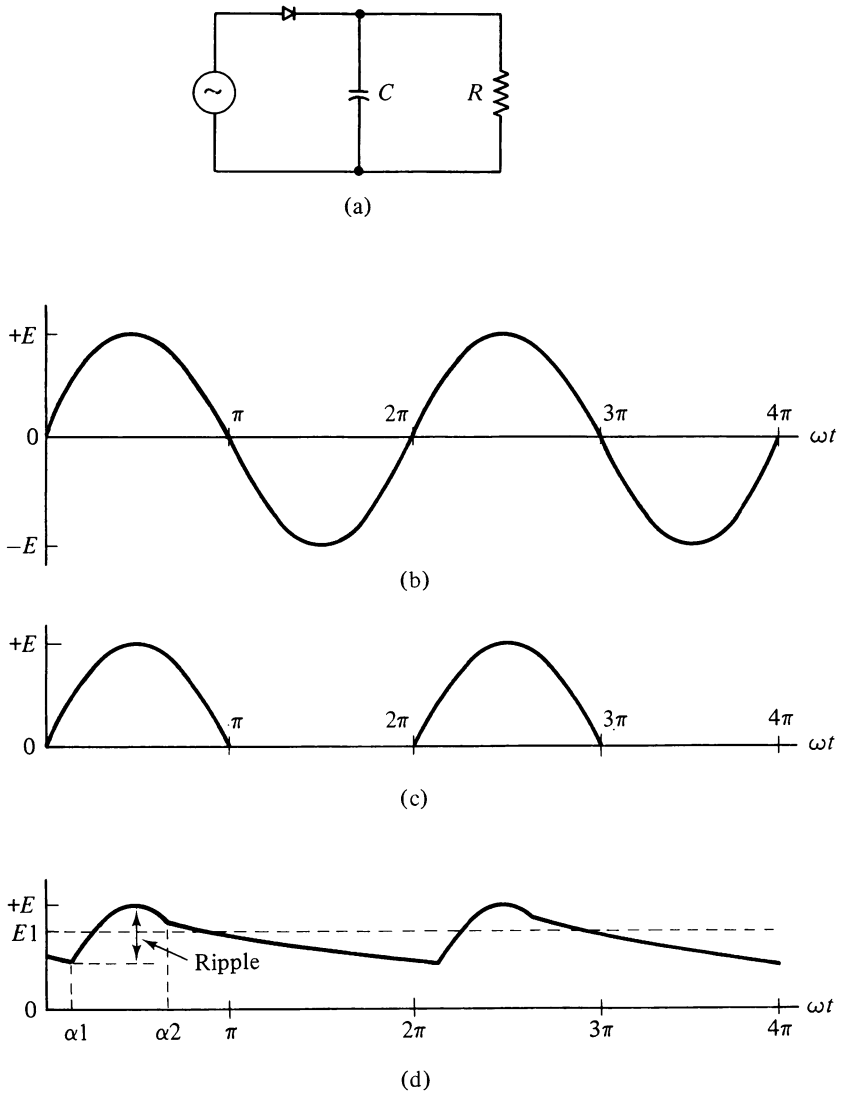


Figure IV-18.

we can calculate $\alpha 1$ (using a root-solving routine) from

$$\sin(\alpha 1) - S \cdot \sin(\alpha 2) = 0$$

and then the average voltage can be determined by integration of the voltage over one complete cycle; the result is

$$E1 = \frac{E}{2\pi} \cdot \left\{ \cos(\alpha 1) - \cos(\alpha 2) + [N \cdot (1 - S)] \right\}$$

Similarly, the average square of the voltage is obtained by integration of the voltage squared over one cycle; the result is

$$\langle E^2 \rangle = \frac{E^2}{8\pi} \cdot \{(2 \cdot \alpha) + \sin(2 \cdot \alpha 1) - \sin(2 \cdot \alpha 2) + [M \cdot (1 - S^2)]\}$$

From physical considerations and a study of Fig. IV-18d, it can be seen that $\alpha 1$ must be between 0 and $\pi/2$ radians. Refer then to Sec. A of Chap. III for a root-solving technique, and try $\omega = 376.991$ rad/sec (which is the appropriate value for normal 60 Hz current, because $\omega = 2\pi f$), $C = 47 \times 10^{-9}$ farads, $R = .33 \times 10^{6}$ ohms, and $E = 300$ volts. You should calculate $\alpha 2 = 1.74018$ radians, $\alpha 1 = .434301$ radians, $E1 = 209.075$ volts, and $RF = .254745$. (*Hint*: Be sure that $\alpha 2$ is in the second quadrant—i.e., between $\pi/2$ and π radians.)

That is quite a large ripple factor, and you'll want to see how much it can be reduced by using a larger capacitor. You should find that subsitution of a capacitance of 1×10^{-6} farads raises the average dc voltage to 293.034 volts and reduces the ripple factor to .0140639, for example.

D-6. Multiple-Feedback Active Filter

The bandpass filter shown in Fig. IV-19 is made up of one active element, the operational amplifier (OP AMP), and five passive elements (three resistors and two capacitors).[10] Suitable values for the components can be determined when the desired center frequency $F0$, the frequency bandwidth B, and the midband voltage gain A have been established. The first step is to calculate Q, where

$$Q = \frac{F0}{B} \tag{IV-65}$$

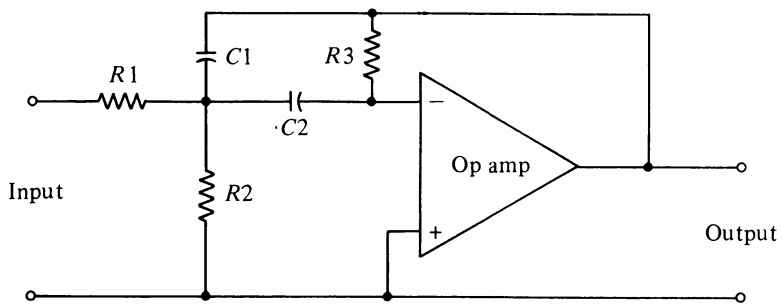


Figure IV-19.

Plan A goes into effect if $Q > \sqrt{|A|/2}$. For this case, $C1$ is chosen to be equal to $C2$ and also equal to a standard capacitance value such that calculated resistances are on the order of thousands of ohms in value. Next solve for $W0$ and H from the following equations

[10] Ferrel G. Stremler, "Simple Arithmetic: An Easy Way to Design Active Bandpass Filters," *Electronics*, (June 7, 1971), pp. 86–89.

$$W0 = 2 \cdot \pi \cdot F0 \tag{IV-66}$$

$$H = \frac{|A|}{Q} \tag{IV-67}$$

Then the following calculations yield values for the resistances:

$$R1 = \frac{1}{H \cdot W0 \cdot C1} \tag{IV-68}$$

$$R = \frac{1}{Q \cdot (C1 + C2) \cdot W0} \tag{IV-69}$$

$$R2 = \frac{R1 \cdot R}{R1 - R} \tag{IV-70}$$

$$R3 = |A| \cdot R1 \cdot \left(1 + \frac{C1}{C2}\right) \tag{IV-71}$$

Plan B goes into effect if $Q \leqslant \sqrt{|A|/2}$. First choose $R1$ equal to 1×10^3 ohms; then calculate $W0$ and H from Eqs. (IV-66) and (IV-67); and then calculate $R3$ from

$$R3 = \frac{H \cdot Q \cdot R1}{1 - \frac{Q}{H}} \tag{IV-72}$$

Next calculate $C1$ and $C2$:

$$C1 = \frac{1}{H \cdot R1 \cdot W0} \tag{IV-73}$$

$$C2 = \frac{H}{R3 \cdot W0} \tag{IV-74}$$

Both of these values should be standardized by rounding them off to the nearest number that is a multiple of 1. (For example, 1.58×10^{-6} farads should be rounded off to 1×10^{-6} farads.) Then the actual component values can be obtained from Eqs. IV-68 through (IV-71) as before.

So design a program that does all this automatically. If Plan A has to be implemented, start with $C1 = C2 = 1 \times 10^{-7}$ farads. Design requirements that exercise both plans are (a) $F0 = 100$ Hz, $B = 16$ Hz, $A = 50$; (b) $F0 = 100$ Hz, $B = 100$ Hz, $A = 100$; and (c) $F0 = 200$ Hz, $B = 40$ Hz, $A = 30$. Partial answers: For (a): $R = 497.359$ ohms, $R3 = 198944$ ohms; for (b): $R = 1575.79$ ohms, $R3 = 160746$ ohms; and for (c): $R = 795.775$ ohms, $R3 = 79577.5$ ohms.

D-7. Linear Network Analysis: First-Order System

A network containing one energy storage element is called a *first-order network* and can be described by an equation of the form[11]

$$\frac{dX}{dT} = (A \cdot X) + U$$

where X is the capacitor voltage if the energy storage element is a capacitor or the inductance current if the energy storage element is an inductor. A is a constant for the network; U is a time-dependent energy source for the network and is therefore called the *forcing function*; and T is the time. The general solution to this equation can be written in the form

$$X(T) = [X(0) \cdot e^{A \cdot T}] + e^{A \cdot T} \int_0^T U(t) \cdot e^{-A \cdot t}\, dt$$

The advantage of this form of the solution is that it clearly separates the present value of X into the contributions of past and current history. The first term represents the value of X when observations are begun (time is zero), and this contribution of the past dies away exponentially (because A turns out to be negative). The second term represents the contribution of the forcing or driving function $U(T)$.

Suppose we specialize our attention to the inductive circuit of Fig. IV-20. Kirchhoff's laws yield

$$L \cdot \frac{dI}{dT} = (-R \cdot I) + V_i(T)$$

or

$$\frac{dI}{dT} = -\left(\frac{R}{L} \cdot I\right) + \frac{V_i(T)}{L}$$

so that the solution for $I(T)$ can be written as

$$I(T) = [I(0) \cdot e^{-R \cdot T/L}] + \frac{e^{-R \cdot T/L}}{L} \int_0^T V_i(t) \cdot e^{+R \cdot t/L}\, dt$$

The current through the inductor can now be obtained for any initial value, $I(0)$, and for any input waveform $V_i(T)$. If you have developed a plotting program (subsection A-5), this will give you a great opportunity to watch the time response of the circuit; for example, the distortion of a square wave input pulse

$$V_i(T) = V_0 \quad \text{for } 0 \leqslant T \leqslant T1$$
$$= 0 \quad \text{otherwise}$$

[11] Benjamin J. Leon and Paul A. Wintz, *Basic Linear Networks for Electrical and Electronic Engineers* (New York: Holt, Rinehart and Winston, Inc., 1970), pp. 329–37.

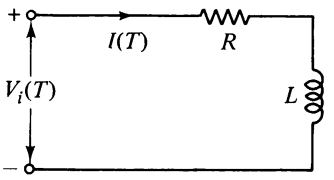


Figure IV-20.

is determined by the relative magnitudes of the circuit time constant (L/R) and the duration of the pulse ($T1$).

As another worthwhile calculation, suppose that $I(0) = 0.5$ ampere, $R = 1.2$ ohms, $L = 5 \times 10^{-3}$ henrys, and the input waveform is sinusoidal:

$$V_i(T) = 7 \cdot \sin(2 \cdot \pi \cdot 60 \cdot T)$$

What is the maximum current through the inductance in the first half-cycle ($T \leqslant 1/120$ second)? With the help of Simpson's Rule (Sec. 3, Chap. III), you can verify a maximum value of 3.76908 amperes.

Or, for the triangular pulse

$$V_i(T) = 700 \cdot T \qquad 0 \leqslant T \leqslant .001 \text{ sec}$$
$$V_i(T) = 1.4 - (700 \cdot T) \qquad .001 \leqslant T \leqslant .002 \text{ sec}$$

what is the current when $T = .002$ sec, assuming $I(0) = 0.8$ amperes, $R = 8.2$ ohms, and $L = 5 \times 10^{-3}$ henrys? (*Answer*: .0639194 amperes.)

D-8. Nonlinear Element Circuit Analysis

The computer is quickly called for circuit analysis when the nonlinear nature of some of its common components is to be considered. (Nonlinear means that the voltage across them is not proportional to the current through them; i.e., their effective resistance is not constant.) Consider the simple diode; rather than a simple on-off switch, it is better described by a logarithmic function:

$$V_D = A \cdot \ln\left(\frac{i_D}{I_0} + 1\right) \tag{IV-75}$$

where V_D is the voltage across the diode, A is directly proportional to the absolute temperature and is about .025 volt at room temperature (300°K), I_0 is the reverse saturation current for the diode, and i_D is the instantaneous current through the diode.

Figure IV-21 shows a relatively simple diode circuit for which we wish to determine the *average* value I_D of the current through the diode.[12] All that needs to be done is to solve simultaneously Eq. IV-75 and the next two equations:

$$E + V_i = V_D + (R3 \cdot I_D) + (R3 \cdot i_D) \tag{IV-76}$$

[12] Donald L. Schilling and Charles Belove, *Electronic Circuits: Discrete and Integrated* (New York: McGraw-Hill Book Company, 1968), pp. 20–37.

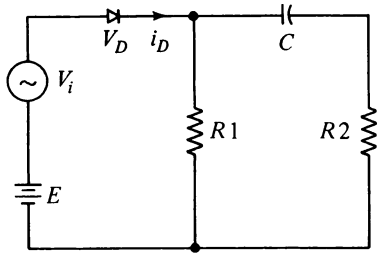


Figure IV-21.

$$I_D = \frac{1}{T} \int_0^T i_D(t)\, dt \tag{IV-77}$$

where E is the constant dc bias voltage and V_i is the ac input voltage. The first equation follows from Kirchhoff's voltage law and the assumption that the capacitor C is sufficiently large that it provides negligible impedance to ac current while blocking dc current. $R3$ and $R4$ are given by

$$R3 = \frac{R1^2}{R1 + R2}$$

$$R4 = \frac{R1 \cdot R2}{R1 + R2}$$

The second equation follows from the definition of I_D as the average diode current; the integration is over one period of the ac voltage ($T = 1/F$).

You can obtain an initial guess for I_D by assuming that i_D has that value when the ac voltage V_i is zero. (This isn't quite correct just because of the nonlinear nature of the diode.) Inserting Eq. IV-75 into Eq. IV-76 then means that you need to start out by solving

$$E = [A \cdot \ln(I_D/I_0 + 1)] + [(R3 + R4) \cdot I_D] \tag{IV-78}$$

for I_D, which obviously requires a root-solving technique (Sec. A, Chap. III). Then the more precise value is obtained by using Eq. IV-77; the procedure is to use this first value of I_D to calculate i_D from Eqs. IV-75 and IV-76 at ten or more equally spaced points within one cycle of the ac voltage and to then use these values of i_D with Simpson's Rule (Sec. E, Chap. III) to obtain another I_D from Eq. IV-77. In general, this last value for I_D is different than the value assumed in solving Eq. IV-76; a new and better guess for I_D is an average of the two. This new guess is used to solve Eq. IV-76 and then Eq. IV-77 again, and the process is continued until convergence to a single value is obtained.

It is of interest to estimate the temperature dependence of I_D; this can be done by multiplying the given value of A by the factor ($T1/300$), where $T1$ is the absolute temperature in degrees Kelvin for the diode.

Suppose then that $I_0 = 1.5 \times 10^{-6}$ amperes, $E1 = 9$ volts, $R1 = 5.6$ ohms, $R2 = 8.2$ ohms, and

$$V_i = 5 \cdot \sin(2 \cdot \pi \cdot F \cdot t)$$

where F is the frequency and is equal to 6000 Hz. For $T1$ = 300°K, you should find that solution of Eq. IV-78 yields I_D = 1.54533 amperes; I_D then converges with a few iterations to a value of 1.5473 amperes. For $T1$ equal to 250°K and 350°K, you should find final average currents of 1.5572 amperes and 1.5375 amperes, respectively.

If you try a saw-tooth waveform,

$$V_i = 5 \cdot F \cdot t$$

you should find an average current of 1.9907 amperes.

E. Physics

E-1. Jumper's Technical Advisor

You have finally been forced to seek gainful employment, and your first job is to make some vital calculations for a man just jumping into the motorcycle-jumping business (Fig. IV-22).

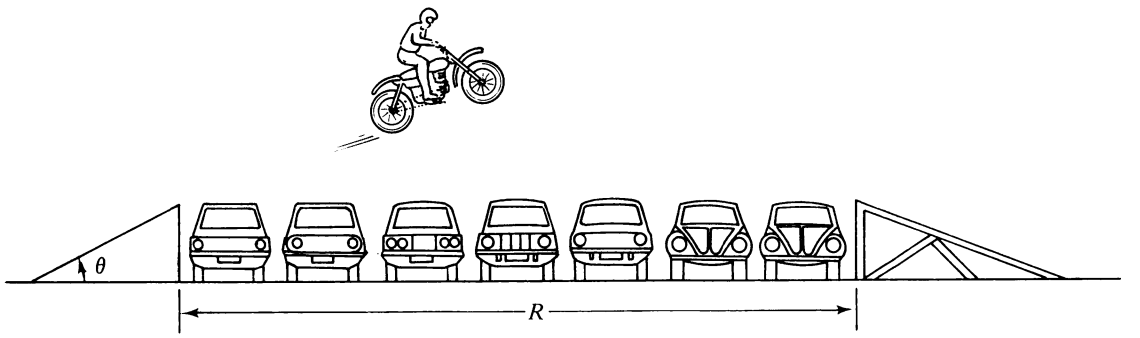


Figure IV-22.

The first obstacle your boss is going to try to hurdle with your help is 22 cars—171 feet worth of distance. You quickly look up the formula for the range of a projectile:

$$\text{Range} = \frac{2 \cdot V^2 \cdot \sin\theta \cdot \cos\theta}{G} = \frac{V^2 \cdot \sin 2\theta}{G}$$

in which the range is given in feet if the angle of the ramp is θ, V is the ramp departure speed in feet per second, and G is the acceleration due to gravity at that spot. You note that an angle of 45° requires the least initial speed, and therefore you whip out your pocket calculator and, using G equal to 32.2 ft/sec^2, suggest to your boss that he use a take-off speed of 74.2 ft/sec (50.6 miles/hr) at θ equal to 45°. He looks at you rather strangely when he hears this and asks how *high* in the air that will put him. You scurry back to your physics textbook and find

$$Y_{max} = \frac{V^2 \cdot \sin^2\theta}{G}$$

and a few more jabs on the pocket calculator allows you to predict a maximum height of 85.5 feet! Your boss informs you in no uncertain terms that his shock absorbers can't take that kind of a drop and you'd better quickly come up with a speed that will reduce the maximum vertical height of 15 feet. You decide that it is time to go to higher-power computing facilities and are soon able to provide the requested calculation. (*A jump of this distance was successfully made in July of 1974.*)

You expect thanks for your work after the successful jump, but instead your boss greets you with some harsh words. "Listen to me, computer man. The only reason I successfully made that jump was because I didn't believe your calculations and went faster than you told me to go! Now you either perfect your theory and correctly predict my actual speed on the last jump or you are out of a job."

So it is back to the drawing board for you. What went wrong? Could the computer have made a mistake? Then it hits you: At these speeds the drag of the air can't be neglected as the textbooks always seem to assume! You search the literature and find that one investigation[13] has shown this drag force to be about

$$\text{Drag force} = .053V^2$$

where the drag is given in pounds if the speed is in feet per second. You don't have the time to do the more accurate analysis of Sec. H of Chap. III, so you assume that this drag force acts only in the horizontal direction; this yields a new expression for the range:

$$\text{Range} = \frac{\ln \{(.004/G) \cdot V^2 \cdot \sin(2\theta)] + 1\}}{.004}$$

where again the range is in feet if the speed is in feet per second. From this you are able to accurately calculate the speed your boss used on his last jump!

The grand finale of your partnership is a well-publicized jump over a 200-foot lake. Your boss says that he will reinforce his shocks as necessary, but he absolutely won't leave the ramp any faster than he did for the last jump. What ramp angle should he use? You rewrite the previous expression as

$$\sin 2\theta = \frac{G \cdot [\exp(.004 \text{ Range}) - 1]}{.004V^2}$$

and are able to provide a suggested angle. What is the speed and the angle? (*In August of 1974, an attempt was made to jump the 200-foot Appalachia Lake in West Virginia; the jump terminated 5 feet short at a reported speed of 95 miles per hour.*)

Answers: 12.68° and 103.5 miles per hour.

E-2. Planetary Mechanics

If the gravitational force of attraction is the only force acting on two bodies, then the sum of their kinetic and gravitational potential energies is constant. The center of mass for the *system* is stationary, because it is assumed that there are no forces outside the system. The pair of bodies are unable to escape each other if their total energy is negative (assuming

[13] Kevin R. Cooper, "Motorcycle Aerodynamics," *Cycle*, Vol. XXVII, No. 9 (September 1976), pp. 65–72.

the usual choice of zero potential energy at an infinite separation distance), and the orbit of one (mass M_2) as viewed from the other (mass M_1) is an ellipse, as shown in Fig. IV-23. For these bound orbits, the total energy is given by[14]

$$W = -\frac{G \cdot M_1 \cdot M_2}{R_P + R_A}$$

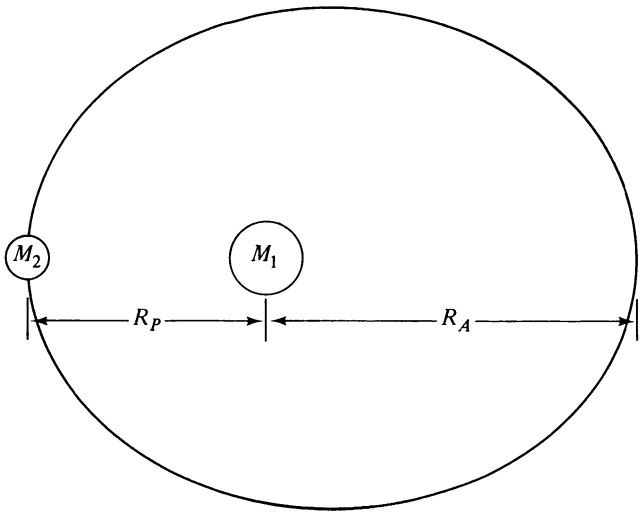


Figure IV-23.

where G is the universal gravitational constant, M_1 and M_2 are the masses of the bodies, R_P is the distance between centers at closest approach (*perigee*), and R_A is the distance between centers at the greatest separation distance (*apogee*). The total energy of the system can then be written as

$$\text{Kinetic energy} + \text{Potential energy} = \text{Total energy}$$

or

$$\left(\frac{\frac{1}{2} M_1 \cdot M_2}{M_1 + M_2}\right) \cdot V_2^2 - \frac{G \cdot M_1 \cdot M_2}{R_2} = \frac{-G \cdot M_1 \cdot M_2}{R_P + R_A}$$

where V_2 is the speed of mass M_2 relative to mass M_1 when it is a distance R_2 from mass M_1. In particular, the relative speed of M_2 at perigee ($R_2 = R_P$) is a maximum and equal to

$$V_P = \left[\frac{2 \cdot G \cdot R_A \cdot (M_1 + M_2)}{R_P \cdot (R_P + R_A)}\right]^{1/2}$$

[14] See, for example, Robert A. Becker, *Introduction to Theoretical Mechanics* (New York: McGraw-Hill Book Company, Inc., 1954), pp. 227–37.

whereas at apogee ($R_2 = R_A$), the relative speed of M_2 is a minimum and equal to

$$V_A = V_P \cdot \left(\frac{R_P}{R_A}\right)^{1/2}$$

The period of the motion is the time that it takes M_2 to make one complete orbit and is given by

$$T = \pi \cdot (R_P + R_A) \cdot \left[\frac{(R_P + R_A)}{2 \cdot G \cdot (M_1 + M_2)}\right]^{1/2}$$

The eccentricity of the orbit can be calculated from

$$e = \left[\frac{-2 \cdot R_P^2 \cdot V_P^2}{G \cdot (R_P + R_A) \cdot (M_1 + M_2)} + 1\right]^{1/2}$$

Consistent units for use in these equations are kilograms for mass, meters for distance, meters per second for speed, and seconds for time. Then a G equal to 6.668×10^{-11} should be used.

A strict check of these equations requires the finding of two isolated bodies, which in turn means considering a universe besides our own. In lieu of this, members of our own solar system should be fairly well described. You can make a quick check of the period equation, for example, by using the moon and the earth. The necessary numbers are $R_A \simeq R_P = 3.84 \times 10^8$ m, $M_{Moon} = 7.35 \times 10^{22}$ Kg, and $M_{Earth} = 5.98 \times 10^{24}$ Kg. You should calculate a period of about 27.2 days, which compares favorably with the accepted lunar sidereal period of 27.3 days.

Moving up in scale, you can determine orbit parameters for the earth-sun system.[15] The mass of the sun is 1.99×10^3 Kg, our closest approach to the sun is 1.471×10^{11} m, and the farthest distance is 1.521×10^{11} m. You should find an orbital speed that varies between 2.99×10^4 m/sec and 3.00×10^4 m/sec, an eccentricity that compares closely with the accepted value of .017, and a period close to the accepted 365.26 days.

Another system, and one in which both masses are really significant, is that of Jupiter and the sun. Here R_P is equal to 7.409×10^{11} m, R_A is equal to 8.157×10^{11} m, and the mass of Jupiter is 1.90×10^{27} Kg. You should calculate a period and an eccentricity close to the accepted values of 11.86 years and .048, respectively.

One more system of great interest is a communications satellite in a synchronous orbit around the earth; this means a circular orbit with a period equal to the rotation period of the earth, 86164.1 seconds (a little less than 24 hours). If the satellite's mass is negligible compared to the earth's mass, as is usually the case, then we can solve the period equation to find the required orbital radius

$$R = \left[\frac{G \cdot M_{Earth} \cdot T^2}{4 \cdot \pi^2}\right]^{1/3}$$

You should find the orbital radius to be 4.217×10^7 m or 26,200 miles from the center of the earth; the orbital speed is 3.075×10^3 m/sec or 6880 miles per hour.

[15] For these and other planetary data, see Carl Sagan, *The Solar System* (San Francisco: W. H. Freeman and Company, 1975), and Elske V. P. Smith and Kenneth C. Jacobs, *Introductory Astronomy and Astrophysics* (Philadelphia: W. B. Saunders Company, 1973).

E-3. Mars Orbit Insertion

A spacecraft approaching a planet such as Mars cannot go into a closed orbit about the planet unless it fires retrorockets to slow down. Fig. IV-24 shows a spacecraft approaching Mars with a speed $V0$ a long distance from the planet; rockets are then fired to reduce the speed to $V1$. If it weren't for the gravitational attraction, it would continue at the speed $V1$ and pass a distance B from Mars, but instead it speeds up, and its path is curved, with A the distance of closest approach. At this point a quick firing of retrorockets can put it into a circular orbit. From considerations of momentum and energy conservation, it can be shown that, for given distances A and B, the speed $V1$ must be

$$V1 = \left[\frac{2 \cdot G \cdot M \cdot A}{B^2 - A^2}\right]^{1/2}$$

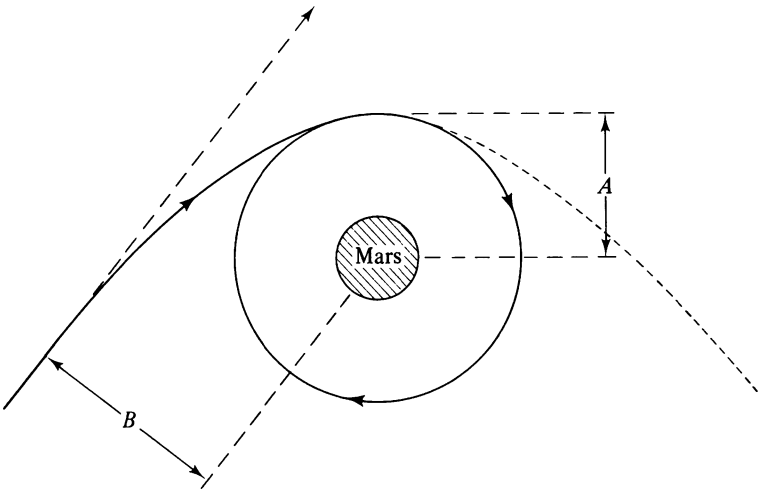


Figure IV-24.

The speed $V2$ when it gets a distance A from Mars is

$$V2 = \frac{G \cdot M}{B \cdot V1} + \left[\left(\frac{G \cdot M}{B \cdot V1}\right)^2 + V1^2\right]^{1/2}$$

and this speed must be reduced to speed $V3$ for a circular orbit:

$$V3 = \left(\frac{G \cdot M}{A}\right)^{1/2}$$

Suppose that B is 6.0×10^6 m and the orbital radius is that used by the Viking orbiter, 4.89×10^6 m. Then, using M equal to 6.4×10^{23} Kg and G equal to 6.668×10^{-11}, you should find $V1$ equal to 5876 m/sec, $V2$ equal to 7209.81 m/sec, and $V3$ equal to 2954.16 m/sec.

A more challenging calculation is involved if we desire a determination of the closest orbital radius for a given amount of fuel. Suppose that the spacecraft is approaching Mars with enough fuel to fire the retrorockets for a total length of time T and that each second of firing reduces the speed by the constant K. Then far from Mars, the speed $V0$ should be reduced to a speed $V1$ given by solving the equation

$$V1 - V0 + V3 - V2 + (K \cdot T) = 0$$

where $V2$ is a function of $V1$ as given in the earlier formula and

$$V3 = \left(\frac{G \cdot M \cdot V2}{B \cdot V1}\right)^{1/2}$$

You can solve this equation for $V1$ by "persistent substitution" or, better, by the root-solving method of Chap. III. Using input values of $V0$ equal to 4.0×10^3 m/sec, K equal to 25 m/sec^2, T equal to 120 seconds, and B equal to 5×10^6 m, you should find a $V1$ such that A is 2.84161×10^6 m and $V3$ is 3875.3 m/sec.

E-4. Mars Landing Mission

Now that you've managed a circular orbit around Mars, you'll want to land for discussions with the natives. Many people have been first attracted to the computer by playing a lunar landing game[16] with it, but at least half the fun is doing the programming in the first place, and there are plenty of other celestial bodies such as Mars on which to practice.

Figure IV-25 shows the geometry of the problem; we find ourselves in a circular orbit around Mars at a height $Y0$ above the surface and with an orbital speed $V0$. We can fire our rockets at any time to exert a force F on the spacecraft at any desired angle θ to the vertical. Our proposed landing spot is presently a distance $X0$ ahead of us, as measured along the curved surface as shown. For this coordinate system, Newton's Second Law yields these equations for the accelerations in the X and Y directions:

$$AX = -\frac{2 \cdot VX \cdot VY}{Y + R} - \frac{F \cdot \sin(\theta)}{M1} \qquad \text{(IV-79)}$$

$$AY = \frac{VX^2}{Y + R} - \frac{G \cdot M2}{(Y + R)^2} + \frac{F \cdot \cos(\theta)}{M1} \qquad \text{(IV-80)}$$

where $M1$ is the mass of the lander (including the fuel), $M2$ is the mass of Mars, R is the radius of Mars, VX is the component of velocity in the X direction (initially equal to $V0$), and VY is the component of velocity in the Y direction (initially equal to zero).

We can make the calculations more manageable by assuming that these accelerations are constant during a short time interval ΔT and by using the standard kinematic equations for constant acceleration; then the positions at the end of the time interval can be determined from

[16] Some examples are to be found in Harold Weinstock, "Cooperative Ventures in Curriculum Development," in Ronald Blum, ed., *Computers in Undergraduate Science Education Conference Proceedings* (Maryland: Commission on College Physics, 1971), pp. 9–22, and in David H. Ahl, ed., *101 BASIC Computer Games* (Massachusetts: Digital Equipment Corporation, 1973).

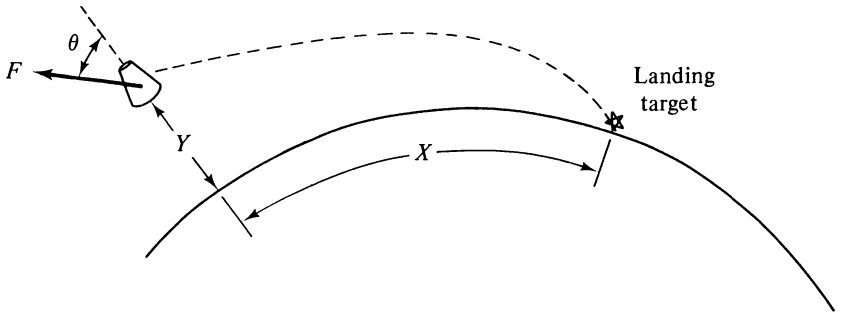


Figure IV-25.

$$X \rightarrow X + (VX \cdot \Delta T) + \frac{AX \cdot \Delta T^2}{2} \tag{IV-81}$$

$$Y \rightarrow Y + (VY \cdot \Delta T) + \frac{AY \cdot \Delta T^2}{2} \tag{IV-82}$$

and the new speeds can be determined from

$$VX \rightarrow VX + (AX \cdot \Delta T) \tag{IV-83}$$

$$VY \rightarrow VY + (AY \cdot \Delta T) \tag{IV-84}$$

Suppose that the fuel that we started with has a mass $F0$; suppose also that in the time interval ΔT a mass of fuel equal to $K \cdot F \cdot \Delta T$ is expelled, where K is the rate of fuel consumption and F is the thrust force. This means that the mass of the fuel has been reduced by

$$F0 \rightarrow F0 - (K \cdot T \cdot \Delta T) \tag{IV-85}$$

as has been the mass of the lander:

$$M1 \rightarrow M1 - (K \cdot T \cdot \Delta T) \tag{IV-86}$$

The time interval ΔT must be chosen so that you can easily use long time intervals initially to slow the ship and to allow gravity to bring it close to the surface, but short time intervals are desirable for the critical phases of the landing approach and touchdown. A good practical compromise that yields a realistic response to thrust force while maintaining a reasonable run time is always to use a ΔT that is one-twentieth of the time duration specified by the pilot (you) for a particular amount and direction of the thrust force.

If we are working in the metric system, then appropriate constants are

$M2$ = mass of Mars = 6.4×10^{23} kg
G = gravitational constant = 6.668×10^{-11}
R = radius of Mars = 3.394×10^{3} m

Try a spacecraft with

$M1$ = mass of lander = 2.5×10^4 kg
$Y0$ = orbital radius = 1.496×10^6 m
K = 7.14×10^{-5} kg N^{-1} sec^{-1}
$F0$ = 2×10^4 kg
$X0$ = 4×10^5 m

The calculations of your landing mission program should then follow this outline:

1. Assign initial values of zero to the coordinates X and Y.
2. Determine the orbital speed from (see previous problem)

$$V0 = \sqrt{\frac{G \cdot M2}{R + Y0}}$$

3. Assign initial values of $V0$ and zero to the speed VX and VY, respectively.
4. Tell the rockets whether to fire or not and, if so, the amount of thrust to use and the duration of this thrust in seconds. The thrust force is in newtons, but the numbers are large enough that you should plan to supply the requested force in *thousands* of newtons and have the program multiply this number by 1000 before using it in the equations. Some maximum value should be established for the available thrust; 400,000 newtons is a reasonable starting point for this kind of lander.
5. Calculate the new values for X, Y, VX, VY, $M1$, and $F0$ after the requested rocket force has been exerted for the requested time; use Eqs. IV-79 through IV-86 for this. While in this loop, you should maintain a continuous check for an overshoot ($X > X0$), a touchdown ($Y \leq 0$), or an emptying of the fuel tanks ($F0 \leq 0$). Print a warning at the end of the loop in the event of an overshoot, but immediately jump out of the time loop and print final values when touchdown occurs; it's all over then. Also, immediately set the thrust force equal to zero as soon as the fuel is gone, but stay in the loop so that the ensuing crash can be adequately evaluated.
6. Print out the latest values for X, Y, VX, VY, $F0$, and the total elapsed time.
7. If you've not landed, go back to step 4 for another decision.
8. If you have landed, decide whether congratulations are in order or whether it wasn't one you could walk away from.

Hint: One successful landing for this Mars lander required 5402 seconds, covered a horizontal distance $X0$ of 400,529 m, had a vertical touchdown speed of .117 m/sec, a horizontal touchdown speed of .037 m/sec, and had 1913 kg of fuel remaining after landing. This kind of precision isn't too difficult to attain if no restriction on thrust is made (other than a maximum for it) and if no restriction is made on the shortness of the time duration. Add some of these restrictions for additional challenge once you get the hang of piloting the ship. This is a natural program for a video display terminal too.

Having acquired the skill to make the intelligent decisions necessary for a soft landing, you are ready for the ultimate sacrifice; because with sufficient persistence and cleverness, you can replace yourself with additional programming instructions and then just stand back and watch the computer make a soft landing to a specified spot. (If you are programming in a noninteractive language, you'll have to start at this point.) The concepts of proportional and derivative control that are discussed in subsection E-9 could be very useful in doing this.

E-5. Barrier Penetration

To put the ball on the other side of a wall, one must give it a speed such that its initial kinetic energy is greater than its initial gravitational potential energy relative to the barrier; then it will reach the height of the barrier with some speed to spare. The result of throwing the ball is thus completely determinate in classical physics—but not so for alpha particles and electrons and other such minute things! Their behavior is described instead by the different set of rules given by quantum mechanics. For them there can be a significant probability that they will appear on the other side of the barrier even for initial energies *less* than the barrier potential energy (Fig. IV-26). This provided an explanation for alpha particle radioactivity and was an early triumph of the quantum theory; it also explains electron tunneling (current is observed going through a thin insulating layer even with a very small applied voltage).

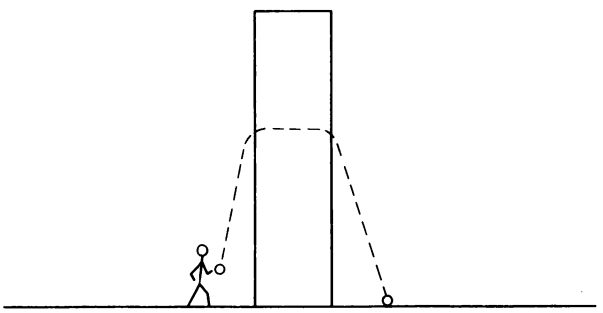


Figure IV-26.

If we define the variable X as the ratio of the particle's kinetic energy to its potential energy, then the quantum mechanical probability of barrier penetration for X less than one (kinetic energy less than potential energy) is given by[17]

$$P = \left\{1 + \frac{[\sinh \sqrt{2 \cdot S \cdot (1 - X)}]^2}{4 \cdot X \cdot (1 - X)}\right\}^{-1} \qquad X < 1$$

where S is a measure of the opaqueness ("thickness") of the barrier.

You can begin an analysis of this strange phenomenon by calculating and printing P as a function of S and X for X equal to 0.1 to 0.9 in steps of 0.1. For a rather opaque barrier ($S = 85.9$), I find that P is equal to 2.27920×10^{-11} for $X = .1$ and equal to 3.61601×10^{-4} for X equal to 0.9.

This relatively thick barrier shows rather interesting behavior just as the kinetic energy increases to values greater than the barrier potential energy (Fig. IV-27). The probability abruptly approaches 100 percent—but then it oscillates to much smaller probabilities as the kinetic energy increases even more! This probability can be calculated from the equation

$$P = \left\{1 + \frac{[\sin \sqrt{2 \cdot S \cdot (X - 1)}]^2}{4 \cdot X \cdot (X - 1)}\right\}^{-1} \qquad X > 1$$

[17] See, for example, Leonard I. Schiff, *Quantum Mechanics*, 2nd ed. (New York: McGraw-Hill Book Company, Inc., 1955), pp. 92–95.

The location of maxima and minima for this function can be found by equating the derivative dP/dX to zero. The result is that the maxima can be located by finding the values of X that satisfy the equation

$$\sin \sqrt{2 \cdot S \cdot (X - 1)} = 0$$

and the minima can be located from the equation

$$\frac{2 \cdot X \cdot (X - 1) \cdot S \cdot \cos \sqrt{2 \cdot S \cdot (X - 1)}}{\sqrt{2 \cdot S \cdot (X - 1)}} - [(2 \cdot X) - 1] \cdot \sin \sqrt{2 \cdot S \cdot (X - 1)} = 0$$

To solve these equations you really need a root-solving algorithm such as that given in Sec. A of Chap. III. For S equal to 85.9, you should find that the first five maxima are at X equal to 1.05745, 1.22979, 1.51703, 1.91917, and 2.43621 and that the first four minima have probabilities equal to .355282, .655346, .825084, and .908658—all of which are consistent with the graph of Fig. IV-27.

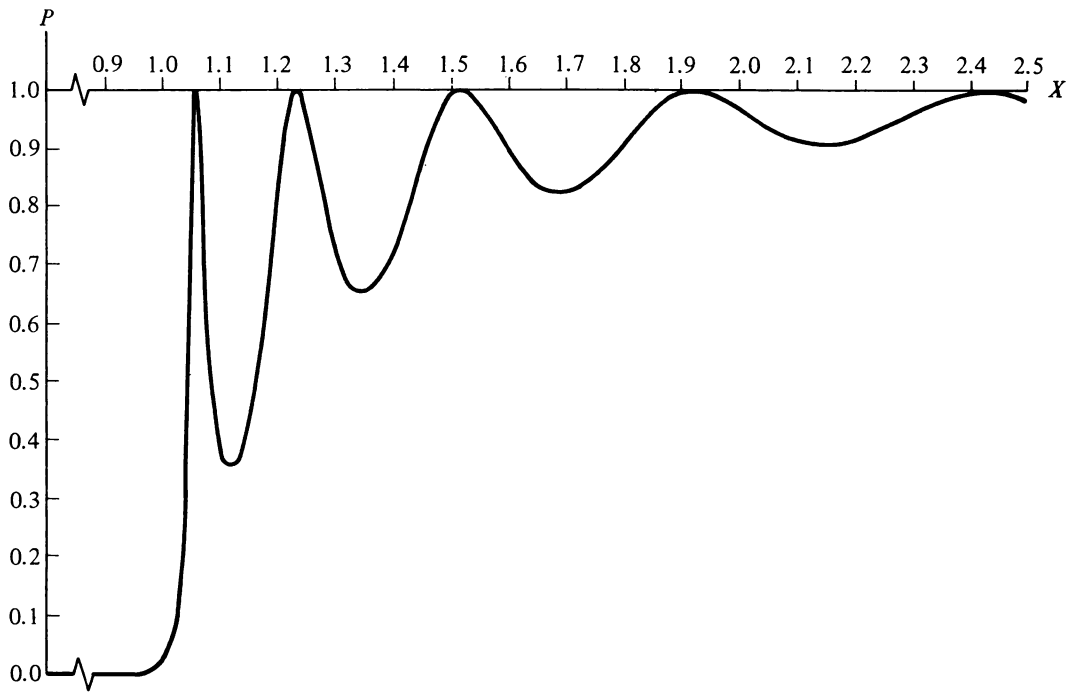


Figure IV-27.

E-6. Real Pendulums

The traditional simple pendulum studied in introductory physics consists of a weight (bob) at the end of a light string (light enough that the weight of the string can be neglected). However, this problem admits to a simple solution only if the maximum value for the angle θ is less than about 7°, for which case the sine of the angle can be approximated by the angle itself, and the resulting formula has a maximum error of about .1 percent. But

consider a real pendulum of length L, with a large amplitude oscillation corresponding to a maximum angle of 60° (Fig. IV-28). The differential equation satisfied by the motion is

$$\frac{d^2\theta}{dT^2} = \frac{G}{L} \cdot \sin\theta$$

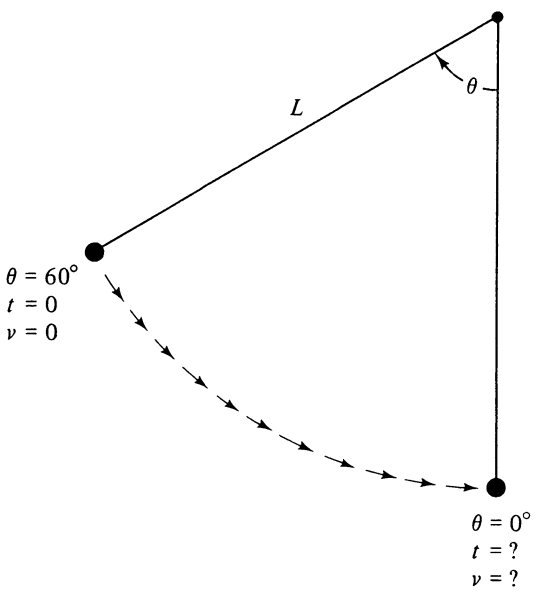


Figure IV-28.

So use the Runge-Kutta second-order method of Sec. G in Chap. III to find the time and speed when the pendulum reaches the bottom of its motion if it is released from a 60° just as timing is begun. Note that the speed at any point is given by the formula

$$V = L \cdot \frac{d\theta}{dT}$$

For L equal to 3 ft and G equal to 32.2 ft/sec^2, I find that it takes .515 seconds to reach the bottom, and the speed there is 9.82852 ft/sec. For comparison, the small angle approximation for the time is $T = (\pi/2) \cdot \sqrt{L/G}$ and yields a value of .47946 seconds. The speed is calculated exactly from energy considerations as $V = \sqrt{2 \cdot G \cdot L \cdot (1 - \cos\theta)}$, and this formula yields the consistent value of 9.82853 ft/sec.

E-7. Underwater Peashooter

The Reynolds number is a predictor of the nature of the fluid flow around a moving object and thereby the resulting fluid drag; it is defined by

$$\text{RN} \equiv \frac{(\text{Density of fluid}) \cdot (\text{Speed of object}) \cdot (\text{Significant dimension of object})}{\text{Viscosity of fluid}}$$

Dust particles and other small, light objects in air have a low Reynolds number. So too do somewhat larger, heavier, and faster objects in a denser fluid such as water. And if the fluid medium is tar, the Reynolds number is almost certain to be low even for the heaviest object! Physically, the criterion is that the viscous retarding force (fluid drag) be of the same order of magnitude or greater than the driving force (which is the weight minus the buoyant force for a falling object).

For these conditions of low Reynolds number, the viscous retarding force can be accurately represented by Stokes' law: The drag force is directly proportional to the instantaneous speed. Also the buoyant force is given by Archimedes' principle: The buoyant force is equal to the weight of the displaced fluid.

Suppose then that an object is projected through a fluid medium, starting with an initial speed $V0$ at an angle θ *below* the horizontal (Fig. IV-29). From Newton's Second Law we can write separately the equations of motion for the X and Y directions:

$$-K \cdot VX = M1 \cdot \frac{dVX}{dT} \tag{IV-87}$$

$$(-K \cdot VY) + (M0 - M1) \cdot G = M1 \cdot \frac{dVY}{dT} \tag{IV-88}$$

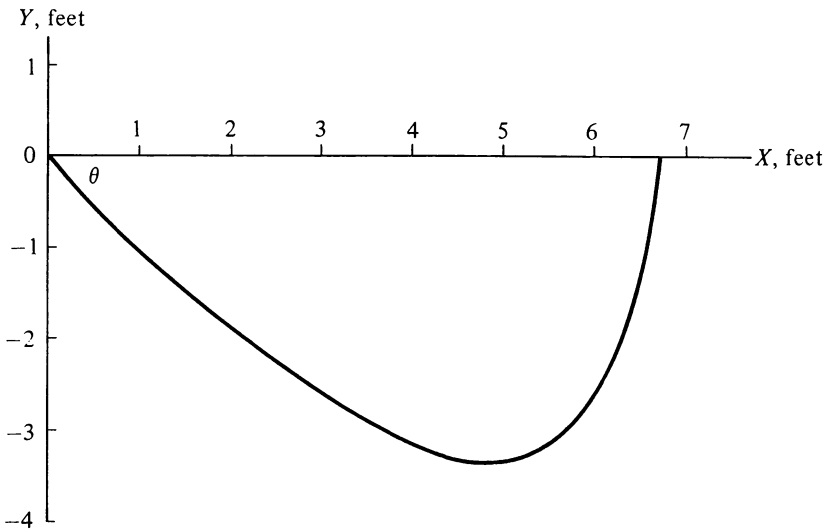


Figure IV-29.

where K is the drag constant, $M1$ is the mass of the object, $M0$ is the mass of the displaced fluid, VX is the speed in the X direction, VY is the speed in the Y direction, and T is the time. (Note that if $M0$ is greater than $M1$, then the buoyant force is greater than the weight of the object, and it will eventually return to its original Y position.) These equations are easily integrated to find the speeds and position of the object at any time T after launch:

$$VX = V0 \cdot \cos(\theta) \cdot e^{-K \cdot T/M1} \tag{IV-89}$$

$$VY = \left\{ \frac{-(M0 - M1) \cdot G}{K} - [V0 \cdot \sin(\theta)] \right\} \cdot e^{-K \cdot T/M1} + \frac{(M0 - M1) \cdot G}{K} \tag{IV-90}$$

$$X = V0 \cdot \cos(\theta) \cdot (1 - e^{-K \cdot T/M1}) \tag{IV-91}$$

$$Y = \left[\frac{-M1 \cdot G \cdot (M0 - M1)}{K^2} - \frac{M1 \cdot V0 \cdot \sin(\theta)}{K}\right] \cdot (1 - e^{-K \cdot T/M1}) + \frac{(M0 - M1) \cdot G \cdot T}{K} \tag{IV-92}$$

The actual speed at any time is the vector sum of the X and Y components:

$$V = \sqrt{VX^2 + VY^2} \tag{IV-93}$$

The time it takes to reach the lowest point, $T1$, may be considerably shorter than the total time of "flight." VY is zero there, so it can be calculated from

$$T1 = \frac{M1}{K} \cdot \ln\left[1 + \frac{V0 \cdot K \cdot \sin(\theta)}{(M0 - M1) \cdot G}\right] \tag{IV-94}$$

and the algorithm of Fig. II-3 could be very useful in this calculation if the second term within the parentheses is very small.

For the special case of a spherical object at low Reynolds number, the drag constant K can be calculated from[18]

$$K = 6 \cdot \pi \cdot R \cdot \mu \tag{IV-95}$$

in which R is the radius of the object and μ is the viscosity of the fluid.

These equations may appear unduly serious to you, but they are actually quite appropriate and easily used to describe an underwater peashooting game. Suggested physical properties are R = .0078 ft (a *small* pea), $M1 = 3.50 \times 10^{-6}$ slugs, $M0 = 3.85 \times 10^{-6}$ slugs, and μ (water) = 2.36×10^{-5} lb-sec/ft^2. Note again that the pea must end up at the surface of the water, because it has less mass than the mass of the displaced water. Suppose that our peashooter has a muzzle velocity $V0$ of 10 ft/sec at an initial angle θ equal to 45°. How long does it take the pea to reach the lowest point in the water? (*Answer*: 1.16601 sec.) What are the position, speed, and elapsed time for the pea when it returns to the level at which it was fired ($Y = 0$) and smashes into the target? (*Answers*: X = 6.72681 ft, V = 2.76711 ft/sec, and T = 3.04866 sec. The very nonparabolic shape of the pea's path is that shown in Fig. IV-29.)

The maximum range is obtained for an angle θ much less than 45°, unlike the traditional projectile in the absence of fluid drag. For this particular pea, you should be able to verify that you'll get the greatest horizontal distance by using an initial angle of about 26.39°, and this will yield a range of about 7.8269 ft and a time of flight of about 2.087 sec.

E-8. Resistance Thermometry

A carbon or germanium resistor is very useful as a secondary thermometer in the temperature range from 0.3°K to 4.2°K. Resistors make good thermometers in that their resistance can be a very sensitive function of the temperature, and resistance can be measured accurately and quickly using either ac or dc techniques. Calibration of the resistor is com-

[18] See, for example, Salamon Eskinazi, *Principles of Fluid Mechanics*, 2nd ed. (Boston: Allyn and Bacon, Inc., 1968), p. 438.

monly made by using a paramagnetic salt at the lower temperatures and a table of He^4 vapor pressure versus temperature at the higher temperatures. But the calibration must necessarily be at a finite number of points, and the measurements are boresome at best. So usually the measured T versus R calibration points are fitted to a function, and that function is used thereafter to obtain temperatures from measured resistances.

It has been found empirically that a good functional form is[19]

$$T = \left[A1 + A2 \cdot \log(R) + \frac{A3}{\log(R)}\right]^{-1}$$

This is a good opportunity to use the nonlinear least squares fitting procedure given in Sec. D of Chap. III.[20] Here are some actual calibration data:

Resistance (ohms)	*Temperature (°K)*
1562.7	2.11745
1590.2	2.05220
1629.1	1.96674
1659.2	1.90589
1693.1	1.84221
1749.3	1.74542
1787.0	1.68702
1821.5	1.63679
1862.2	1.58208
1899.8	1.53490
1935.6	1.49261
1981.3	1.44273
2023.3	1.40008
2062.7	1.36271
2105.0	1.32512

Hint: $A1$ is about -40, $A2$ is about 3, and $A3$ is about 125. You should be able to get a χ of about 0.006 or less.

E-9. Computer as Controller

More and more the computer is playing an active role in our lives: Preprogrammed microcomputers are in record players, televisions, microwave ovens, radios, cars, and aircraft. In the laboratory the minicomputer has become a familiar face because it can be programmed to take data, process data, and make experimental decisions based on the processed data. This "real-time" use of computers to interact with and control an experiment while it is taking place can be illustrated with the problem of creating a temperature controller.

In the previous problem, a resistance thermometer was calibrated. Suppose that we have placed it in thermal contact with an experimental chamber that is surrounded by cold, liquefied gases (Fig. IV-30). The chamber is connected both mechanically and thermally to a liquid He^3 reservoir whose temperature is held at .35°K. The liquid outside this region is

[19] J. R. Clement and E. H. Quinnell, "The Low Temperature Characteristics of Carbon-Composition Thermometers," *Rev. Sci. Instru.*, Vol. 23, No. 5 (May 1952), pp. 213–16.

[20] A fitting procedure that seems to work better in practice is given by R. E. Harris, *Fortran Programs for Cryogenic Thermometry*, University of Illinois Department of Physics Technical Report No. 1, January 1966.

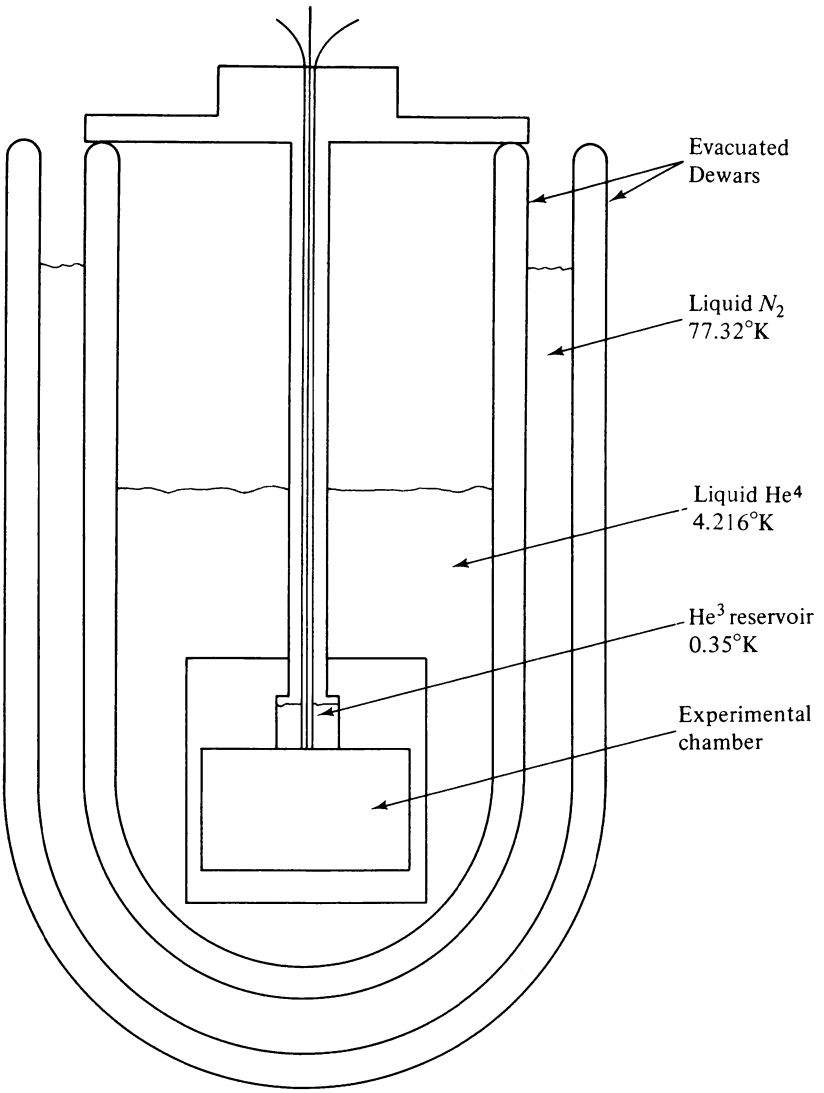


Figure IV-30.

liquid He^4 and is quietly boiling at 4.216°K; buffering this liquid against the hot outside world is a liquid bath of nitrogen at 77.32°K.

The experimental chamber is thermally coupled to the He^3 and also to room temperature (273°K) through the electrical and mechanical connections; the result is that it will stabilize at some temperature above 0.35°K. We've wrapped fine copper wire around the experimental chamber as a heater wire so that we can obtain experimental temperatures above this equilibrium temperature. To reach and stabilize the chamber at a higher temperature, it might seem that all we need to do is provide power to the heater wire in direct proportion to the differences between the present temperature and the desired temperature. However, the chamber often has sufficient internal thermal resistance that it takes a few seconds or longer for the temperature to reach its equilibrium value; this is analogous to the time lag in the electrical charging of a capacitor through a resistor, so we also characterize it by a time constant τ.

The first task is to model the response of the system to heat input. Suppose that the system currently is at a temperature T and that the equilibrium temperature for the system in the absence of heater current is $T0$. Suppose also that there is a certain amount of extra heat energy in the system at the beginning of a time interval Δt. This heat energy by itself tends to increase the temperature, but the system is a dynamic one; thus the heat won't immediately change the system temperature as measured by our thermometer because of thermal lag, and throughout the time interval there is a heat loss to the low-temperature reservoir and a heat gain from the surroundings and mechanical connections. A simple model for the response of the system is:

$$\Delta T = K \cdot \left[Q \cdot (1 - e^{-\Delta t/\tau}) + \left(1 - \frac{T}{T0}\right) \right] \cdot \Delta t$$

ΔT here is the temperature change during the time interval Δt. Q is proportional to the heat capacity and to the extra heat energy in the system at the start of the time interval. K is a constant that depends on such things as the thermal coupling to the system. Note that this model is reasonable: if Q is zero, then the temperature doesn't change when T equals $T0$; if there is a nonzero heat energy added, the temperature increases; all the heat is not effective in changing the temperature because of the time constant in the exponential, but, if the system time constant is very short compared to the sampling interval, then the exponential is nearly zero, and its effect is negligible.

Suppose we start with a system in equilibrium at temperature $T0$. When we supply heat, Q is positive and the temperature increases. But a fraction of this Q, equal to $Q \cdot [\exp(-\Delta t/\tau)]$, is still on its way through the system at the end of the time interval. Thus the Q for the second interval must include this leftover amount as well as any additional amount that we add at that time. This running total for Q must be maintained for each time incrementation.

The next task is to devise a program that intelligently supplies Q to the system. If we have simple proportional control, the leftover Q will make the system overshoot the desired temperature. So we also need a control term that is dependent on the *rate* at which the temperature is approaching the equilibrium point. For this we need to keep track of the temperature difference at the beginning of the previous time interval ($T1 - T_{\text{old}}$) and compare it with the temperature difference at the beginning of the present time interval ($T1 - T$), where $T1$ is the desired temperature; the difference between these temperature differences is the basis for a derivative control term. Finally, we need a continuous supply of Q even when we reach the desired temperature, because it is higher in temperature than the equilibrium value. A simple way to obtain this contribution is to average the past three amounts that were *added* to Q. Thus our computer program must tell a current controller to provide an amount of Q to the system given by

$$\Delta Q = [A \cdot (T - T1) + B \cdot (T - T_{\text{old}})] + \tfrac{1}{3}(Q1 + Q2 + Q3)$$

where A determines the relative strength of the proportional term, B determines the relative strength of the derivative term, T is the current system temperature as measured by the thermometer, $T1$ is the desired temperature, T_{old} is the temperature at the beginning of the previous time interval, and $Q1$, $Q2$, and $Q3$ are the amounts added to Q in the previous three time intervals. [The calculated amount of current goes to a digital to analog converter (DAC) first.]

Your task then is to write a program based on these equations that simulates both the experimental chamber and the controller and to determine optimum values for A and B (i.e., a critically damped system). If B is too large, the Q that is provided from this last formula will oscillate in value, and the time to reach the desired temperature will be excessively long (overdamped system). If B is too small, the Q that is provided will cause an overshoot and an oscillation about the desired temperature (underdamped system). Also, B needs to be larger as the system time constant gets larger relative to the sampling interval.

For a short time constant (τ = .1 sec., Δt = .5 sec.), I find that critical damping brings the simulated system to within one millidegree (.001°K) of a desired temperature of .57°K in about 12 sec. (Use k = .27 and $T0$ = .38°K and start with A equal to 1; $Q1$, $Q2$, and $Q3$ should all start at zero.) For a long time constant (τ = 2.5 sec., Δt = .5 sec.), I find that with critical damping, about 14 sec. are needed to get within one millidegree.

APPENDICES

APPENDIX 1

BASIC Operators, Functions, Commands

A. Symbols Used for Mathematical Operations

Symbol	*Meaning*	*Examples*
–	Negates a constant or a variable	–4, –A
–	Subtraction operator	7 – 3, A – B
+	Addition operator	7 + 3, A + B
*	Multiplication operator	7*3, A*B
/	Division operator	7/3, A/B
↑	Exponentiation operator ($7\uparrow 3 \equiv 7^3$)	7↑3, A↑B
( )	Separates two operators; orders the operations	–(–7), A*(–B) 3*(7+3), A*(A+B)

NOTE: A few terminals use the symbol ^ in place of the symbol ↑ .

B. Priority of Operations

The priority of operations is (a) exponentiation, (b) multiplication and division, and (c) addition and subtraction. A BASIC expression is evaluated from left to right in the absence of parentheses and with operators of equal priority. With parentheses the expressions are evaluated from the innermost pair outward. Here are some examples.

BASIC Expression	*Algebraic Equivalent*
X = A + B*C↑D – E	$X = A + (B \cdot C^D) - E$
X = A↑B↑C	$X = A^{(B^C)}$
X = A*B – C↑D/E + F	$X = (A \cdot B) - \frac{C^D}{E} + F$
X = A/B↑3	$X = \frac{A}{B^3}$
X = (A/B)↑3	$X = \left(\frac{A}{B}\right)^3$
X = (A + 2*(B + 3)/C)*D	$\left[A + \frac{2 \cdot (B + 3)}{C}\right] \cdot D$

C. Relational Operators

Relational operators are used in an IF. . . THEN statement to compare two BASIC expressions and then perform a given operation based on the result of the comparison.

Symbol	*Meaning*	*Example*
=	Equal to	`10 IF A=B THEN C=A/2`
<	Less than	`10 IF A<B THEN GO TO 20`
>	Greater than	`10 IF A>B THEN STOP`
> =	Greater than or equal to	`10 IF A>=B THEN GOSUB 1000`
< =	Less than or equal to	`10 IF A<=B THEN A=0`
< >	Not equal to	`10 IF A<>B THEN A=B`

D. Built-in Mathematical Functions

Functional Form	*Resulting Operation*	*Example of Use*
SQR(A)	The square root of the present value of A is obtained, assuming A is equal to or greater than zero. A here and in the following may be a numeric constant, a variable, or any valid BASIC expression.	`10 A=SQR(A+3*B)`
SIN(A)	Obtains the sine of the angle A assuming that A is in radians.	`10 A=SIN(3*B)`
COS(A)	Obtains the cosine of the angle A, assuming that A is in radians.	`10 A = COS(3)`
ATN(A)	Obtains the angle in radians whose tangent is A. If A is positive, then the angle will be in the first quadrant (0 to $\pi/2$ radians); if A is negative, then the angle will be in the fourth quadrant (0 to $-\pi/2$ radians).	`10 A = ATN(B/2)`
ABS(A)	Obtains the absolute value of A.	`10 A = ABS(B-2)`
EXP(A)	Obtains the value of e^A, where $e = 2.71828....$	`10 A = B*EXP(B)`
LOG(A)	Obtains the natural logarithm of A; A must be positive.	`10 A = LOG(B*3)`
INT(A)	Obtains the greatest integer less than or equal to A.	`10 A = INT(B-A)`
RND(A)	Obtains a random number between 0 and 1, independent of the value of A.	`10 A = RND(4)`
SGN(A)	Returns 1 if A is positive, 0 if A is zero, and -1 if A is negative.	`10 A = A*SGN(B)`

NOTE: Many BASIC interpreters recognize additional mathematical functions such as PI(π) and TAN(A) (tangent of A). Matrix operators and functions are also available on the larger systems.

E. Basic BASIC Statements

Key Word(s)	*Use*	*Example*
LET	Assign a value to a variable. (The word LET is often optional.)	`10 LET A = SQR(B)`
PRINT	Print the value of a variable or variables. Print text if the text is enclosed in quotes.	`10 PRINT A,4*B,C/2` `20 PRINT "D=" D`
READ, DATA	Assign a value to a variable or variables.	`10 READ X,Y` `20 DATA 3,4`
INPUT	Assign a value to a variable or variables from number(s) typed on the terminal keyboard during execution.	`10 INPUT X,Y`
RESTORE	Makes the interpreter go back to the beginning of the DATA statement(s) for value(s) of variable(s) in a READ statement.	`50 RESTORE`
GO TO	Transfer execution to a different statement line.	`10 GO TO 7`
IF . . THEN	Conditionally transfer execution or perform an operation.	`10 IF A<0 THEN A=-A`

FOR . . NEXT	Repeatedly execute a set of statements, incrementing a variable each time.	`10 FOR A=1 TO 3 STEP .5`
		`20 B = SIN(A)`
		`30 PRINT A,B`
		`40 NEXT A`

STOP	Stops program execution.	`70 IF A<0 THEN STOP`
REM	Provides information for programmer; ignored by the interpreter during execution.	`10 REM ROOT PROGRAM`
RANDOMIZE	Starts the "random" number generator at a random point in its sequence of numbers.	`80 RANDOMIZE`
DIM	Reserves storage locations for subscripted variables.	`10 DIM A(10),B(20)`
DEF FNa (A,B)	Defines a function that will be evaluated during execution much as the built-in functions. Some systems allow only one variable. "a" here is any letter of the alphabet.	`10 DEF FNX(A)=4*SQR(A)`
GO SUB	Direct execution to a subroutine.	`30 GO SUB 1000`
RETURN	Returns execution from the subroutine to the statement immediately following the GO SUB statement.	`1050 RETURN`
END	Last statement of a program (often optional).	`1100 END`

F. Keyboard Commands

Key(s) or Word	*Meaning*	*Printed Example*
RUBOUT	Erase the last typed character(s); ← is printed for each keystroke.	10 X-Y←←=Y
RETURN	Returns the carriage to the left-most position and signals the end of a statement or command.	(NOTHING IS PRINTED)
(Any number)	Deletes the statement with that statement number.	10
BYE	Terminates user/system interaction.	BYE
NEW	Clear program memory; names following program.	NEW BLITZ
OLD	Retrieves a previously stored program.	OLD BLITZ
SAVE	Saves the current program; it is stored on a designated system device if not otherwise specified.	SAVE
CTRL/C (keys together)	Stops program execution (the form of	↑C
or ESC	the command varies from system	↑P
or CTRL/P, etc.	to system).	
CTRL/U (keys together)	Erases all of the current line (the form	↑U
or SHIFT/L, etc.	of the command varies from	
	system to system).	\
RUN	Begins program execution.	RUN
RUNNH	In some systems, this starts program execution without printing first a header line with the program name.	RUNNH
LIST	Lists the program currently in the memory.	LIST
LIST 10	Lists only the statement on line number 10. (The commands to list specified *groups* of statements are not well standardized.)	LIST 10

APPENDIX 2

Multi-User BASIC

A multi-user system is one in which the central processor of the computer rapidly alternates its attention among two or more terminals and can appear (to the users) to be simultaneously executing each of their programs. A sample program is given here for RT-11 BASIC on Digital Equipment Corporation's PDP-11 minicomputer. It is assumed that the program BLITZ has previously been stored on the system device (which could be a hard disk, a floppy disk, a tape, etc.). The characters that are underlined are those that the programmer typed on the keyboard of the terminal.

```
PLEASE SAY HELLO

HELLO
USERID:  WH
PASSWORD:  (typed but not printed)

MU BASIC/RT-11 ON THE AIR!

READY

OLD BLITZ

READY

LIST

BLITZ     4-APRIL-84          MU BASIC/RT-11          V01-01

10 REM BLITZ IS THE NAME

20 PRINT "CHEERS"
```

```
30 END

READY

RUNNH

CHEERS

BYE

USER WH IS LOGGED OFF.
```

APPENDIX 3

BASIC Error Messages

The format of the error messages varies considerably from computer to computer, although the common mistakes of programmers do not. The message may be printed out in full, abbreviated, or given as a code number. If the error occurs during execution, then the line number at which the difficulty occurs is also printed. Some errors are "fatal" and terminate execution immediately. Other errors result in a printed message, but execution is continued; for example, when a computation results in a number smaller than the smallest that the interpreter can store (less than 10^{-38} in some systems), the interpreter may call it zero and go on.

Here are some common execution error messages in DEC's PDP-11 Multi-User BASIC.

Message	*Meaning*
?ARG	Argument error: The BASIC interpreter has encountered a series of characters that isn't in its vocabulary.
?BDR	Bad DATA read: There is an illegal character in a DATA statement.
?BLG	Bad log: The argument for the LOG function is zero or negative.
?FAD	Function already defined: More than one definition of a user-defined function.
?FOV	Floating overflow: The result of a computation is a number larger than that which can be stored.
?FSV	FOR statements with same variable: Two FOR statements using the same control variable are not separated by a NEXT statement to close the loop of the first one.
?FUN	Floating underflow: The result of a computation is a number smaller than that which can be stored.
?FWN	FOR without NEXT: During execution a FOR statement was found, but it wasn't followed later by a NEXT statement with the same named variable.
?FZD	Floating zero divide: Zero was a divisor in a statement of the program. (Use the immediate mode to ask which variable was responsible.)
?IDM	Illegal DIM: Either the subscript for a variable in a DIMension statement is not an integer or the same variable has been listed twice.
?NGS	Negative square root: A negative number was the argument for the SQR function.

Message	Meaning
?NWF	NEXT without FOR: The interpreter has encountered a NEXT statement before reaching a FOR statement with the same named variable.
?OOD	Out of data: More values for variables have been requested in READ statement(s) than there are numbers in the DATA statement(s).
?RWG	RETURN without GO SUB: The interpreter has encountered a RETURN statement before encountering a GO SUB statement.
?SOB	Subscript out of bounds: The subscript for a subscripted variable is less than zero or greater than its DIMensioned maximum value.
?SYN	Syntax error: The BASIC interpreter has encountered a series of characters that isn't in its vocabulary.
?UFN	Undefined function: A statement attempts to use a user-defined function that hasn't been defined.
?ULN	Undefined line number: A statement number that doesn't exist has been used in another statement (such as a GO TO statement).

APPENDIX 4

Flowcharts and Programs for Chap. II

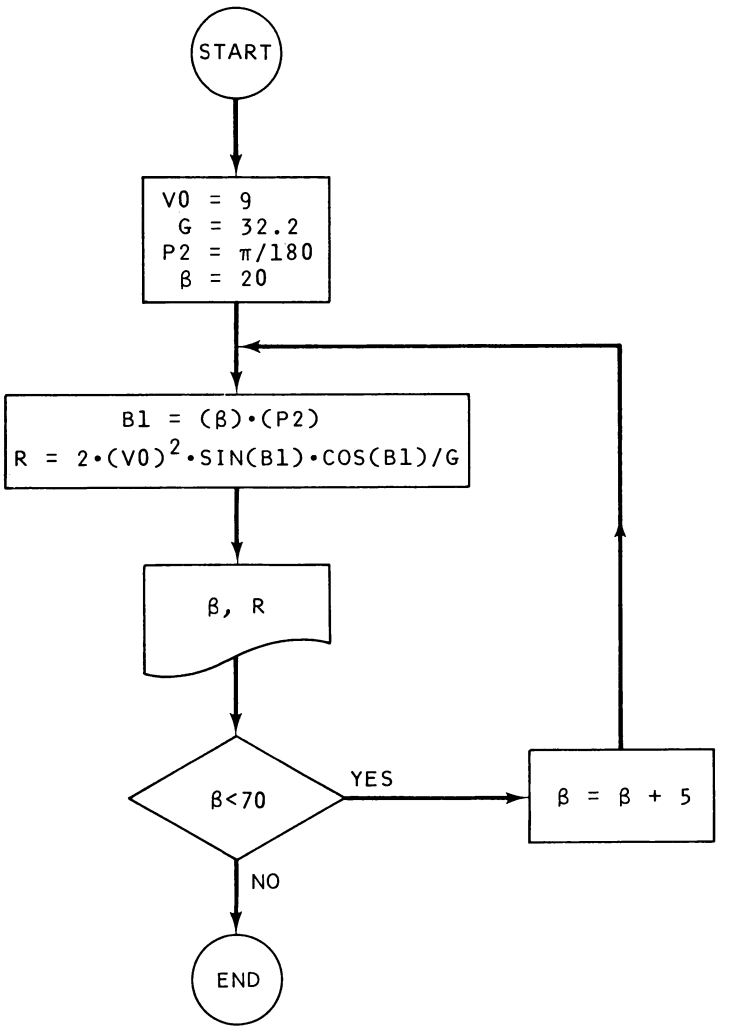


Figure A-1. Flowchart for prob. 2, Chap. 1, Sec. F

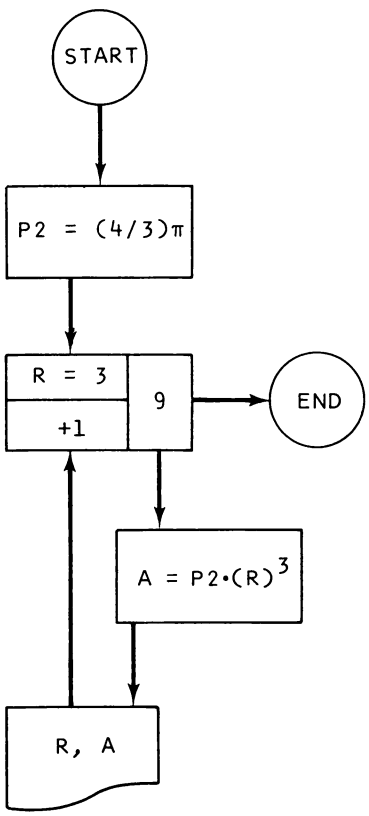


Figure A-2. Flowchart for prob. 3, Chap. 1, Sec. G

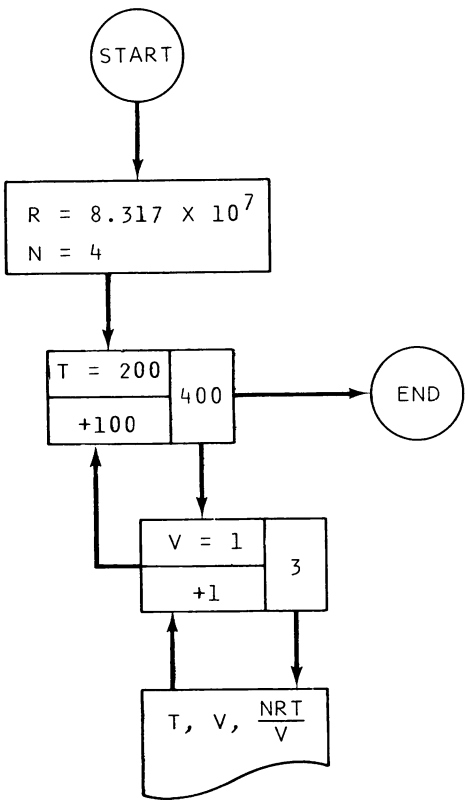


Figure A-3. Flowchart for prob. 4, Chap. 1, Sec. G

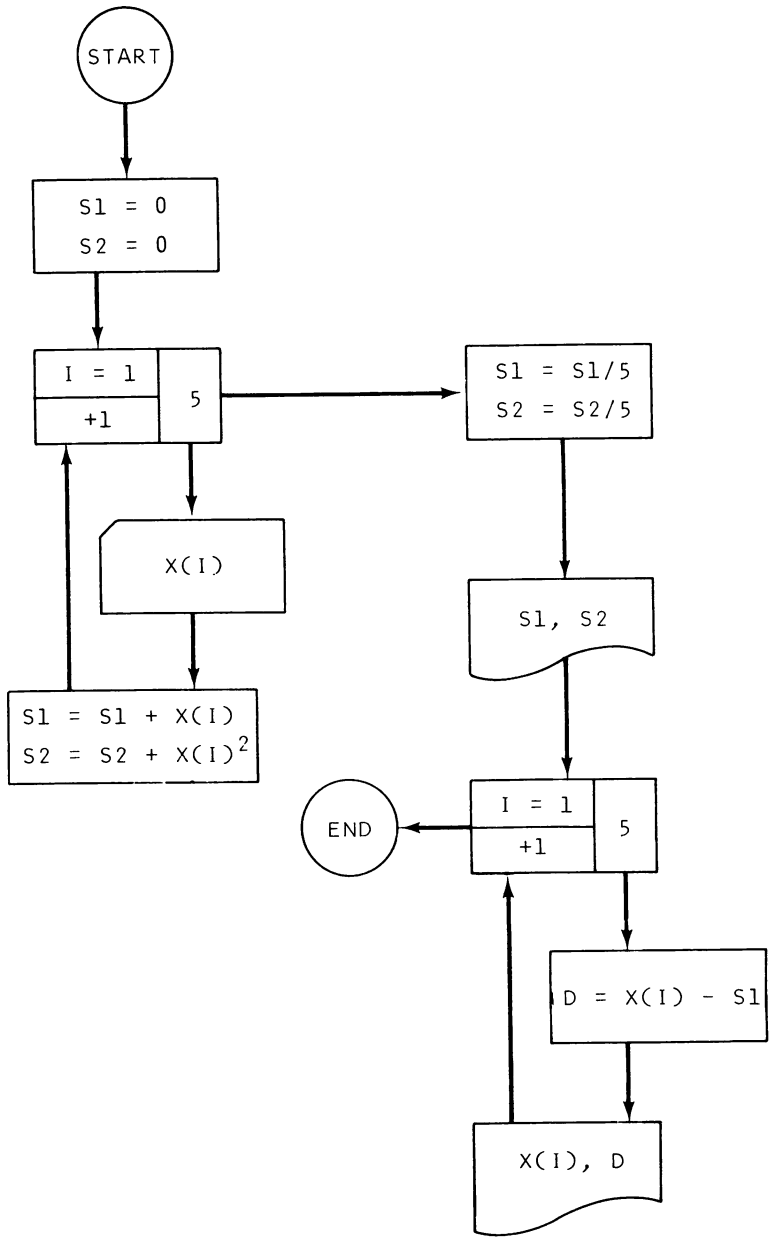


Figure A-4. Flowchart for prob. 2, Chap. 1, Sec. H

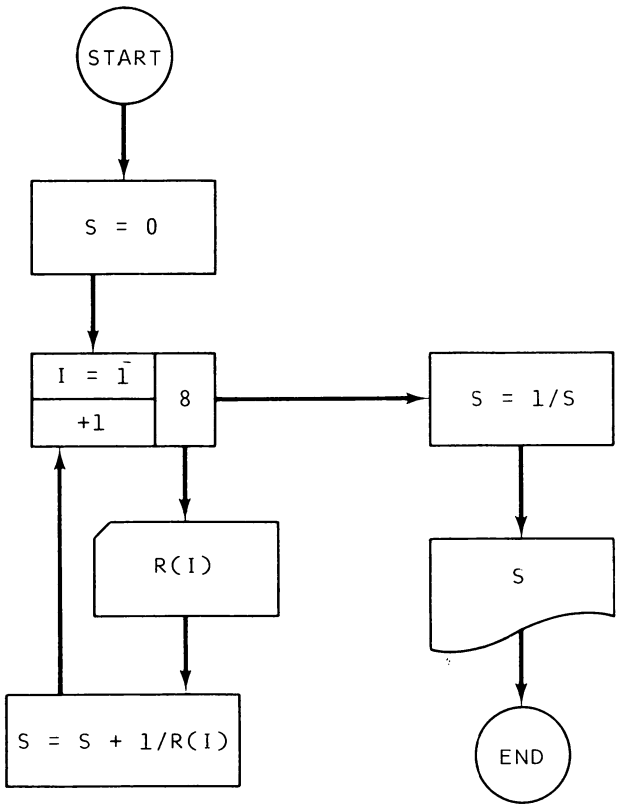


Figure A-5. Flowchart for prob. 3, Chap. 1, Sec. H

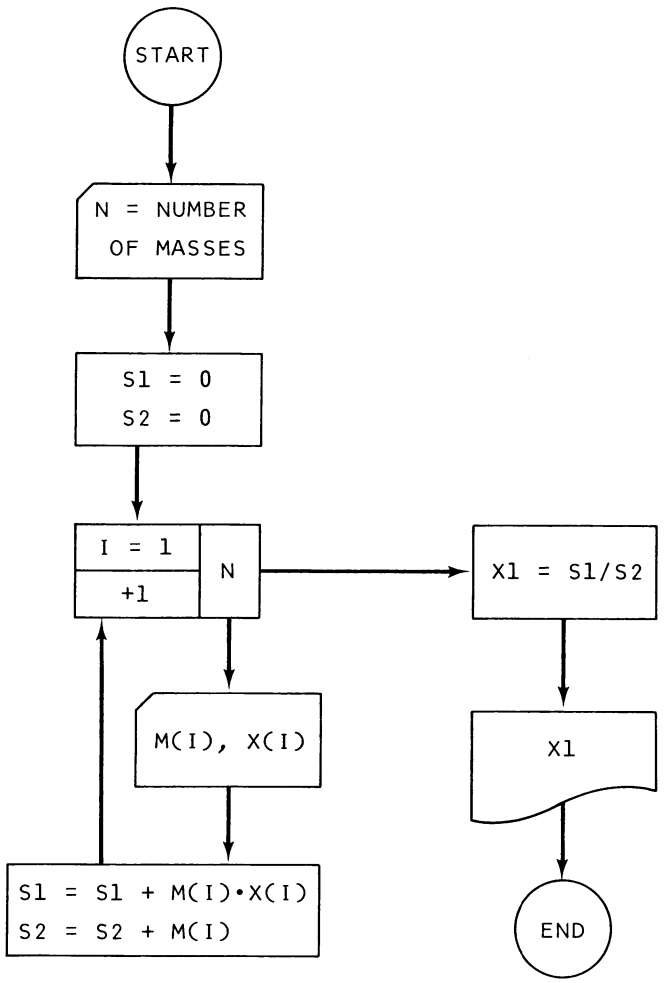


Figure A-6. Flowchart for prob. 4, Chap. 1, Sec. H

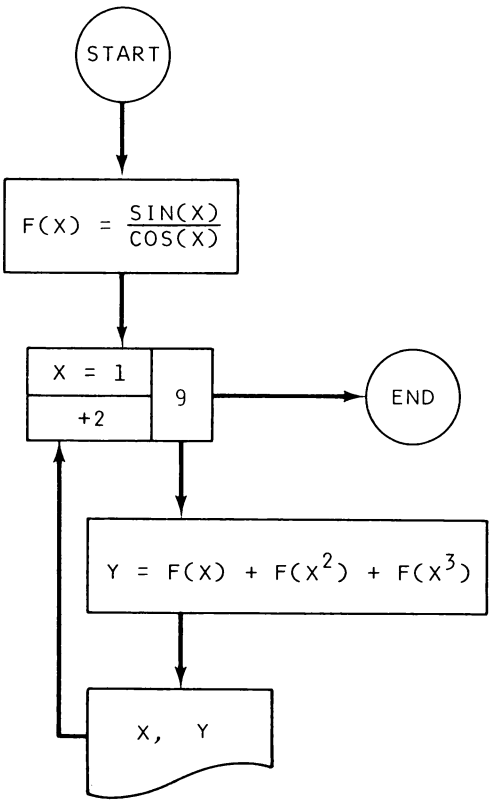


Figure A-7. Flowchart for prob. 1, Chap. 1, Sec. I

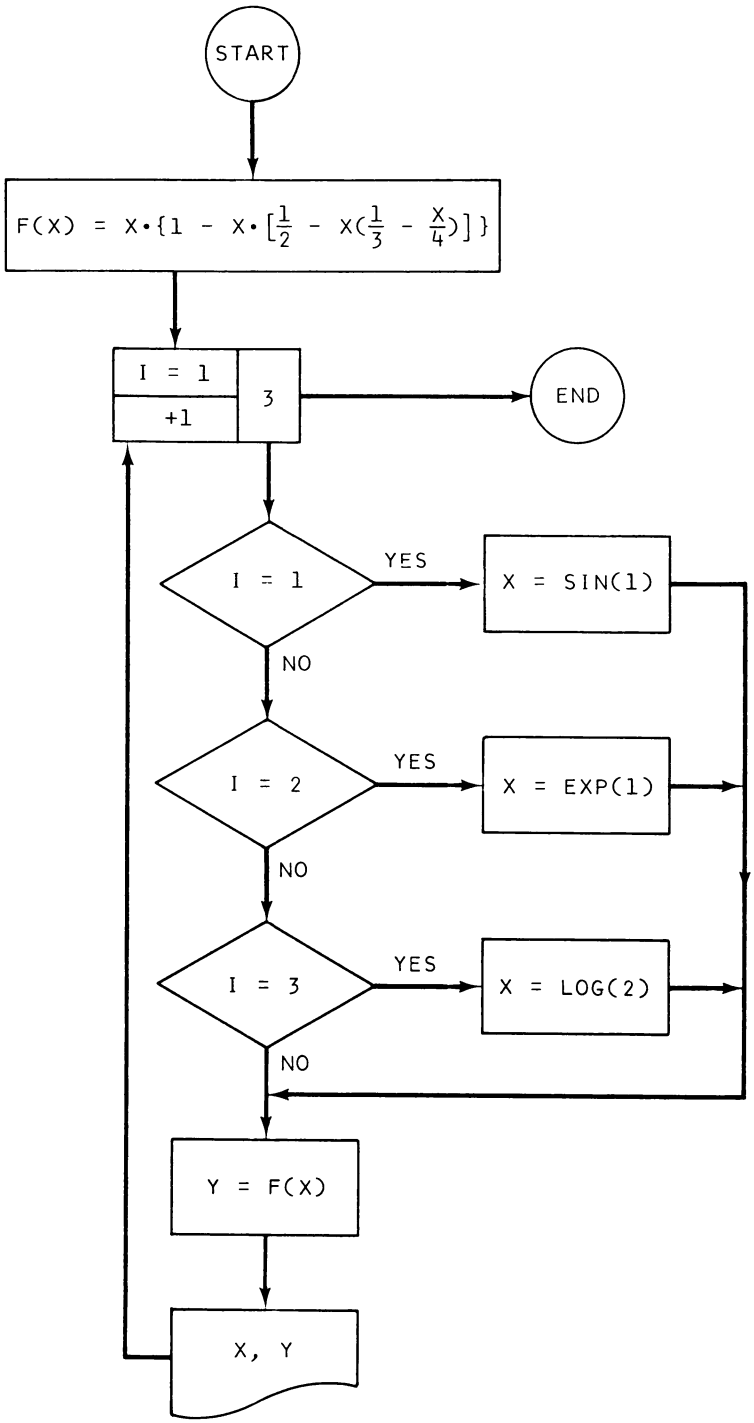


Figure A-8. Flowchart for prob. 4, Chap. 1, Sec. I

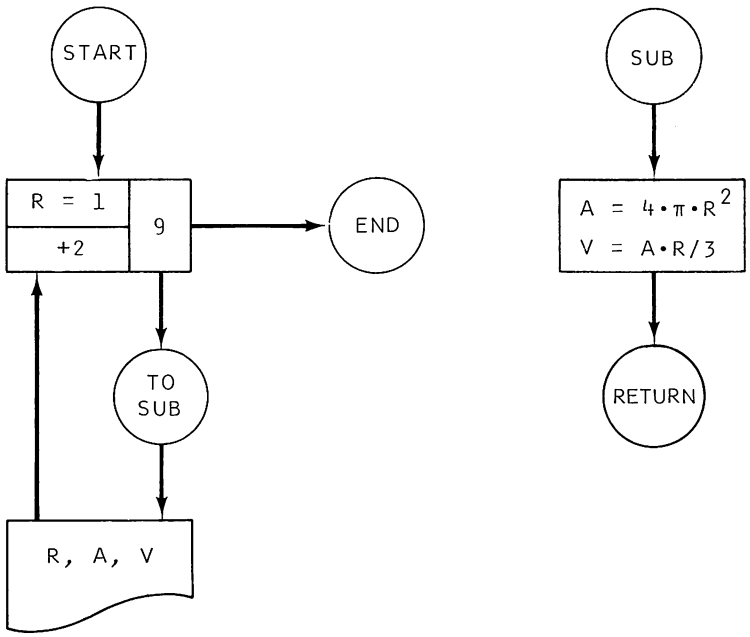


Figure A-9. Flowchart for prob. 3, Chap. 1, Sec. J

BASIC Program

```
10 REM AIRCRAFT LOADING PROBLEM, BASIC SOLUTION PROGRAM
20 W=3170
30 M=106200
40 PRINT "WEIGHTS OF PILOT, FRONT PASSENGER,  BACK PASSENGERS=";
50 INPUT P0,P1,P2
60 W=W+P0+P1+P2
70 M=M+37.06*(P0+P1)+70.98*P2
80 W1=0
90 PRINT "NO. OF ITEMS OF BAGGAGE=";
100 INPUT N
110 FOR I=1 TO N
120 PRINT "WEIGHT OF ITEM(" I; ")=";
130 INPUT W0
140 W1=W1+W0
150 IF W1<=200 THEN GO TO 180
160 PRINT "BAGGAGE IS TOO HEAVY:" W1
170 STOP
180 NEXT I
190 W=W+W1
200 M=M+96.11*W1
210 GO SUB 500
220 IF 4830-W<600 THEN GO TO 260
230 F=600
240 W=W+600
250 GO TO 280
260 F=4830-W
270 W=4830
280 M=M+35*F
290 PRINT "FUEL(GAL)=" F/6
300 GO SUB 500
```

```
310 PRINT "GO FLY!"
320 STOP
500 IF M>41.38*W+2140 THEN GO TO 580
510 IF W<3900 THEN GO TO 550
520 IF M<58.3*W-102000 THEN GO TO 560
530 RETURN
540 IF M<31.8*W+1350 THEN GO TO 560
550 RETURN
560 PRINT "M IS TOO SMALL: M=" M
570 STOP
580 PRINT "M IS TOO LARGE: M=" M
590 STOP
600 END
```

FORTRAN Program

```
C       AIRCRAFT LOADING PROBLEM, FORTRAN IV SOLUTION PROGRAM
        REAL M
        W=3170.
        M=106200.
        READ(5,10) P0,P1,P2
10      FORMAT(3F10.0)
        W=W+P0+P1+P2
        M=M+37.06*(P0+P1)+70.98*P2
        W1=0.
        READ(5,20) N
20      FORMAT(I2)
        DO 60 I=1,N
        READ(5,30) W0
30      FORMAT(F10.0)
        W1=W1+W0
        IF(W1-200.) 60,60,40
40      WRITE(6,50) W1
```

```
50    FORMAT(" BAGGAGE IS TOO HEAVY...",F10.0)
      STOP
60    CONTINUE
      W=W+W1
      M=M+96.11*W1
      CALL CALC(M,W)
      IF(4230.-W) 70,80,80
70    F=4830.-W
      W=4830.
      GO TO 90
80    F=600.
      W=W+600.
90    M=M+35.00*F
      F1=F/6.
      WRITE(6,100) F1
100   FORMAT(" FUEL(GAL)= ",F10. )
      CALL CALC(M,W)
      WRITE(6,110)
110   FORMAT(" GO FLY...")
      STOP
      END
      SUBROUTINE CALC(M,W)
      REAL M
      Q=M-41.38*W-2140.
      IF(Q) 30,30,10
10    WRITE(6,20) M
20    FORMAT(" M IS TOO LARGE. M=", F10.0)
      STOP
30    IF(W-3900.) 40,50,50
40    Q=M-31.8*W-1350.
      IF(Q) 60,80,80
50    Q=M-58.3*W+102000.
      IF(Q) 60,80,80
```

```
60      WRITE(6,70) M
70      FORMAT(" M IS TOO SMALL. M=",F10.0)
        STOP
80      RETURN
        END
```

Index

BASIC Programming For Scientists and Engineers

Wilbert N. Hubin

Here is a practical way for those working in engineering and scientific fields to learn the main functions of computers and how to use them in their work.

Employing the BASIC computer language for programming, the book first describes the fundamentals of BASIC and supplies short practice problems based on common scientific formulas. It points out the most useful features of the language and identifies pitfalls that often confuse beginners.

The author then explains the flowchart method for clarifying and documenting the logic of a computer program. He defines the basic symbolism and rationale of flowcharting and discusses various numerical analysis techniques using flowcharts. The methods presented are easy-to-understand but quite efficient for a wide range of scientific applications.

The last section offers a variety of problems with which to develop programming proficiency. They are physically interesting but simple in concept, and they demonstrate many tasks the computer can do in scientific and engineering areas. Some require numerical analysis methods, many do not.

Representative answers are furnished for all the problems in this book, making it a functional self-training tool for scientists, engineers, and technicians in today's industries.

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

0-13-06648(